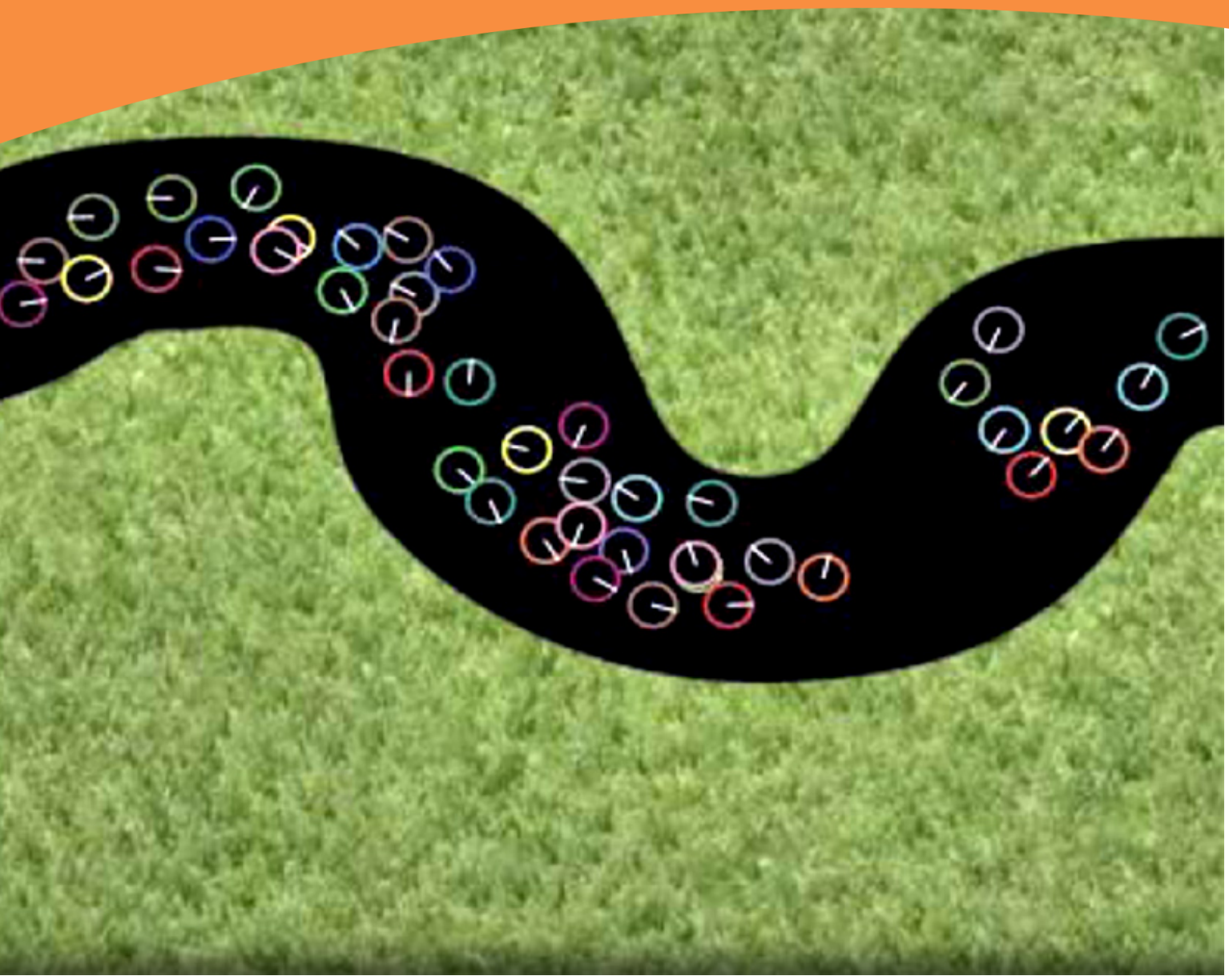
Artificial Intelligence: Exercises I

Agents and Environments

William John Teahan



bookboon

William Teahan

Exercises for Artificial Intelligence

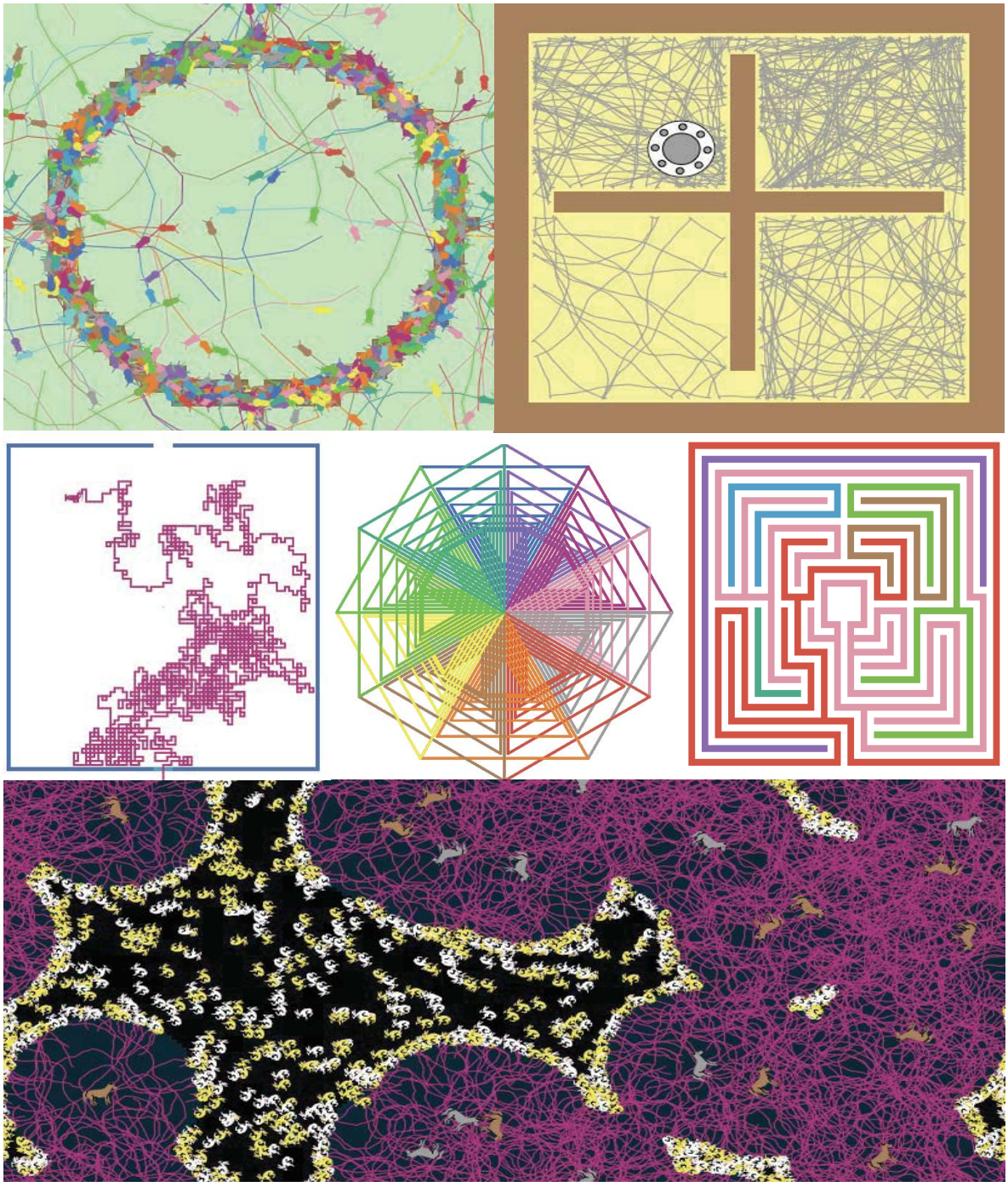
Agents and Environments

Exercises for Artificial Intelligence – Agents and Environments
1st edition

ISBN 978-87-7681-591-2

Contents

Exercises for Artificial Intelligence – Agents and Environments



Selection of screenshots taken from NetLogo models described in this book.

Preface

The list of exercises, chapter headings and section, and NetLogo models in this book closely follow what is in the companion "*Artificial Intelligence – Agent Behaviour I*" book. The best way to learn about what is written in the companion book is to try out each of the NetLogo models that are described in the book and in the exercises below. An index of the models used in these books can be found using the following URL:

NetLogo Models for Artificial Intelligence	http://files.bookboon.com/ai/index.html

A table listing all the models described in this book and the companion book is also provided below. Each entry in the table lists the name of the model, the exercises where it is described, a short description of the model, and a URL where it can be found. Each of these models have sections in the Information tab that provide various documentation, such as: what the model is; how it works; how to use it; the meaning of each of the Interface's buttons, sliders, switches, choosers, monitors, plots and output; important things to notice; things to try out; suggestions for extending the model; explanations of interesting NetLogo features used in the model; credits and references; and links to related models. In particular, the sections on how to use it, things to notice and things to try out provide some suggestions on various things a user can try when playing with the models.

The reader, however, should not restrict themselves to just these suggestions. Due to the complex system nature of many of the simulations that result from the running of these NetLogo models, often unforeseen phenomena emerge as a result of the agent – agent and agent – environment interactions. The reader is encouraged to become an 'explorer' of the virtual environments created by these models by trying out as many of the different combinations of the slider, switch and chooser values as possible while running the simulations many times to ensure that a representative sampling of the possible system behaviours is observed.

Agents Animation (4.5.5, Solution to 4.5.5) This model performs a simple animation of various turtle agent shapes to give the impression that they are flowing past the observer. http://files.bookboon.com/ai/Agent-Animation.html
Ants (5.4.1) This model simulates a colony of ants foraging for food. In NetLogo's Models Library: Biology > Ants. http://ccl.northwestern.edu/netlogo/models/Ants
Chevening House Maze (3.7.2) This model draws a schematic representation of the Chevening House garden maze. http://files.bookboon.com/ai/Chevening-House-Maze.html

Chevening House Maze with Coloured Islands (3.7.4, Solution to 3.7.4)

This model colours the islands in the Chevening House garden maze.
http://files.bookboon.com/ai/Chevening-House-Maze-with-Coloured-Islands.html

Chevening House Maze with Wall Following (3.7.2, Solution to 3.7.2)

This model gets a turtle to wander around the Chevening House maze using wall following behaviour.
http://files.bookboon.com/ai/Chevening-House-Maze-with-Wall-Following.html

Climate Change (3.5.4)

This is a model of energy flow in the earth and simulates climate change due to the presence of CO2 and clouds, for example. In NetLogo's Model Library: Earth Science > Climate Change.
http://ccl.northwestern.edu/netlogo/models/ClimateChange

Continental Divide (4.6.1)

This model animates one method for finding the continental divide. In NetLogo's Model Library: Earth Science > Continental Divide. http://ccl.northwestern.edu/netlogo/models/ContinentalDivide

Empty Maze (3.6.7)

This model draws an empty maze with no inside walls. http://files.bookboon.com/ai/Empty-Maze.html

Empty Maze with Wall Following (3.6.7, Solution to 3.6.7)

This model gets a turtle to wander around the empty maze using wall following behaviour.
http://files.bookboon.com/ai/Empty-Maze-with-Wall-Following.html

Follow Trail (5.5.2, Solution to 5.5.2, 5.5.3, Solution to 5.5.3)

This model allows the user to test out various trail following behaviours for ants. It is an extension of the Santa Fe Trail model. http://files.bookboon.com/ai/Follow-Trail.html

Foxes and Rabbits (3.6.2)

This model creates 100 foxes and 1000 rabbits. http://files.bookboon.com/ai/Foxes-and-Rabbits.html

Foxes and Rabbits 2 (4.2.7, Solution to 4.2.7)

This model creates foxes and rabbits. Once created, the rabbits move away from the foxes if they are too near.
http://files.bookboon.com/ai/Foxes-and-Rabbits-2.html

Grand Canyon (4.6.2)

This model simulates rainfall in part of the Grand Canyon. In NetLogo's Model Library: Earth Science > Grand Canyon.
http://ccl.northwestern.edu/netlogo/models/GrandCanyon

Hampton Court Maze (3.7.1)

This model draws a schematic representation of the Hampton Court Palace garden maze.
http://files.bookboon.com/ai/Hampton-Court-Maze.html

Hampton Court Maze with Turtle (4.2.6, Solution to 4.2.6)

This model gets a turtle to wander around the start of the Hampton Court maze using simple commands.
http://files.bookboon.com/ai/Hampton-Court-Maze-with-Turtle.html

Hampton Court Maze with Wall Following (3.7.1, Solution to 3.7.1)

This model gets a turtle to wander around the Hampton Court maze using wall following behaviour.
http://files.bookboon.com/ai/Hampton-Court-Maze-with-Wall-Following.html

Hatch Example (3.6.3)

This model demonstrates the use of the `hatch` command to simulate turtles reproducing and dying.
http://ccl.northwestern.edu/netlogo/models/HatchExample

Hill Climbing Example 2 (5.3.1) This model show how to give turtle agents a sense of what's up and what's down to perform hill climbing. In NetLogo Model's Library: Code Examples > Hill Climbing Example. See modified code at: http://files.bookboon.com/ai/Hill-Climbing-Example-2.html
Hill Climbing with Wall Following (5.3.1, Solution for 5.3.1) This model implements turtle agents that can use a sense of what's up or down to perform hill climbing, or use a sense of touch via proximity detection to perform wall following, or can do both. http://files.bookboon.com/ai/Hill-Climbing-with-Wall-Following.html
Life Cycle Stages (4.4.2) This model shows an example of finite state automata (FSA) that represents the life cycle stages of people throughout their lives. http://files.bookboon.com/ai/Life-Cycle-Stages.html
Life Example (3.6.3, 3.6.4) This model shows how to use some simple commands in NetLogo to simulate the life cycle of people. http://files.bookboon.com/ai/Life-Example.html
Line of Sight Example 2 (5.3.1) This model shows how to provide turtles with a rudimentary sense of vision based on simulating a line of sight. In NetLogo Model's Library: Code Examples > Line of Sight Example. See modified code at: http://files.bookboon.com/ai/Line-of-Sight-Example-2.html
Load File (3.5.5, 3.5.6, Solution to 3.5.5 & 3.5.6) This model shows how to load text from a file. http://files.bookboon.com/ai/Load-Text.html
Look Ahead Example 2 (5.3.1) This model shows how to provide turtles with a rudimentary sense analogous to the sense of vision. In NetLogo Model's Library: Code Examples > Look Ahead Example. See modified code at: http://files.bookboon.com/ai/Look-Ahead-Example-2.html
Mazes (5.4.2) This model shows how to get a simple reactive turtle agent to move around a maze. http://files.bookboon.com/ai/Mazes.html
Mazes-2 (5.4.4, Solution for 5.4.4) This extends the Mazes model by adding the Butterfly Maze, and two further behaviours based on those from the Searching Mazes model. http://files.bookboon.com/ai/Mazes-2.html
N Dimensional Space (2.6.1, 3.6.5, 3.6.6) This model visualises N dimensional data concerning New Zealand All Blacks. http://files.bookboon.com/ai/N-Dimensional-Space.html
Nested Squares (3.6.8, Solution to 3.6.8) This model provides a solution to Exercise 3.6.8. It draws nested squares six different ways. http://files.bookboon.com/ai/Nested-Squares.html
Nested Triangles (4.2.3, 4.2.4, 4.2.5, Solutions for 4.2.3, 4.2.4, 4.2.5) This model provides solutions to Exercises 4.2.3, 4.2.4 and 4.2.5. It can be used to draw elaborate patterns made out of equilateral triangles. http://files.bookboon.com/ai/Nested-Triangles.html

NZ Birds (4.4.4) This model constructs and animates a decision tree for the problem of identifying New Zealand birds. http://files.bookboon.com/ai/NZ-Birds.html
Santa Fe Trail (3.6.12, 4.3.1, 5.5.1, Solution to 3.6.12, 4.3.1, 5.5.1, 5.5.3 & 5.5.4) This model tests out various behaviours as solutions to the Santa Fe Ant Trail problem. http://files.bookboon.com/ai/Santa-Fe-Trail.html
Shape Animation Example (4.5.1) This model demonstrates how to do basic animation using shapes. In NetLogo's Models Library: Code Examples > Shape Animation Example. http://ccl.northwestern.edu/netlogo/models/ShapeAnimationExample
Shuffle Cards (3.6.10, Solution to 3.6.10) This model shuffles a pack of cards. http://files.bookboon.com/ai/Shuffle-Cards.html
Shuffle and Deal Cards (3.6.10, Solution to 3.6.10) This model shuffles and deals a pack of cards. http://files.bookboon.com/ai/Shuffle-and-Deal-Cards.html
Simple Walk (4.2.1, 4.2.2, Solutions for 4.2.1 and 4.2.2) This model provides solutions to Exercises 4.2.1 and 4.2.2. It gets a turtle to execute some simple walking commands. http://files.bookboon.com/ai/Simple-Walk.html
Stick Figure Animation (4.5.3, 4.5.4) Users of this model can create their own stick figure animations and save them as QuickTime movie files. http://files.bookboon.com/ai/Stick-Figure-Animation.html
Stick Figure Walking (4.5.2) This model provides a simple animation of a stick figure walking. http://files.bookboon.com/ai/Stick-Figure-Walking.html
Sudoku Builder (3.6.11, Solution to 3.6.11) This model allows the user to fill in a Sudoku puzzle. http://files.bookboon.com/ai/Sudoku-Builder.html
Termites (3.4.2, 3.5.1, 3.5.1) This model simulates termites creating piles of wood chips. In NetLogo's Models Library: Biology > Termites. http://ccl.northwestern.edu/netlogo/models/Termites
Termites (Perspective Demo) (3.5.3) This model is a modified version of the Termites model that makes use of NetLogo's perspective features. In NetLogo's Models Library: Perspective Demos > Termites (Perspective Demo). http://ccl.northwestern.edu/netlogo/models/Termites(PerspectiveDemo)
Tumor (3.4.6) This model illustrates the growth of a tumour and how it resists chemical treatment. In NetLogo's Models Library: Biology > Tumor. http://ccl.northwestern.edu/netlogo/models/Tumor
Two States (4.4.1) This model shows how to draw a simple two-state Finite State Automata (FSA) that represents turning a light switch off or on. http://files.bookboon.com/ai/Two-States.html
Vacuum Cleaner Robot (5.3.2, Solution to 5.3.2) This model simulates the actions of a vacuum cleaner robot. http://files.bookboon.com/ai/Vacuum-Cleaner-Robot.html

Vision Cone Example 2 (5.3.1) This model shows how to provide turtles with a rudimentary sense of vision. In NetLogo Model's Library: Code Examples > Vision Cone Example. See modified code at: http://files.bookboon.com/ai/Vision-Cone-Example-2.html
Wall Following Example (5.3.1) This model shows how to provide turtles with an ability to sense and follow nearby walls. In NetLogo Model's Library: Code Examples > Wall Following Example http://ccl.northwestern.edu/netlogo/models/WallFollowingExample
Wolf Sheep Predation (3.4.5) This model simulates predation of sheep by wolves. In NetLogo's Models Library: Biology > Wolf Sheep Predation. http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

1 Introduction

1.1 What is "Artificial Intelligence"?

Exercise 1.1.1:

One of the issues for Artificial Intelligence research concerns the problem of categorization for intelligent systems – the problem of how to select suitable categories to cover a set of examples, and the resultant classification errors that arise once a particular set of categories has been chosen.

As an example, use your favourite search engine to find out how the term 'Artificial Intelligence' is defined by different people. How is the definition different to the one provided in the first sentence of Section 1.1 of the companion book *Artificial Intelligence – Agents and Environments*? Classify the various definitions you find into a set of different categories. Which definitions fit well into your set of categories? Which definitions pose problems that require a re-evaluation of the suitability of your taxonomical classification?

1.2 Paths to Artificial Intelligence

Exercise 1.2.1: Computer Science Unplugged Activity – The Turing Test

Computer Science Unplugged (Bell, Witten & Fellows, 1998) is a series of learning activities designed to teach important concepts from Computer Science without the use of computers. Download the Computer Science Unplugged Activity concerning the Turing Test from http://csunplugged.org/turing-test and follow the section on 'What to do' in order to try out an unplugged variation of the test for intelligence.

How successful were the participants in this activity at finding out who the 'computer' was? In light of the insights you gained from this activity, do you now believe or not believe the Turing Test is a valid test for intelligence? Explain why.

Exercise 1.2.2:

The person playing the 'computer' for this activity must reply to specific questions using pre-determined answers. Several of these questions and answers have been reproduced in the following table. (For the full set of questions and answers, follow the URL listed in Exercise 1.2.1).

Question	Answer
What is the name of Bart Simpson's baby sister?	I can't remember.
Are you a computer?	Are you a computer?
What is 2 × 78?	166 (*This is deliberately incorrect!*)
What is the square root of two?	1.41421356237309504878
Add 34957 to 70764.	*Wait for about 20 seconds before giving the answer…*105621.
Do you like school?	Yes, I like school.
For which country is the flag a red circle on a white background?	I don't know.
What food do you like to eat?	I'm not hungry, thanks.

Table 1.2.2. Some responses for the 'computer' in the Computer Science Unplugged activity concerning the Turing Test.

Do you think these are good questions to ask during the Turing Test as a test for intelligence? Do you think that if you were one of the judges you would be fooled by these answers? If not, which answers would give the 'computer' away?

Exercise 1.2.3: Computer Science Unplugged – The Turing Test

In the description of the Turing Test unplugged activity mentioned in Exercise 1.2.1 (on page 223 of the Computer Science Unplugged book) the following is stated:

"*No artificial intelligence system has been created that comes anywhere near passing the full Turing test. Even if one did, many philosophers have argued that the test does not really measure what most people mean by intelligence. What it tests is behavioral equivalence: it is designed to determine whether a particular computer program exhibits the symptoms of intellect, which may not be the same thing as genuinely possessing intelligence. Can you be humanly intelligent without being aware, knowing yourself, being conscious, being capable of feeling self-consciousness, experiencing love, being…alive*?"

Do you agree or disagree with these remarks? Provide a full explanation of your point of view.

Exercise 1.2.3:

Do you think the Turing Test is a valid test for Artificial Intelligence? What types of intelligent human behaviour are covered by the Turing Test? Which types would best be served by different tests?

1.3 Objections to Artificial Intelligence

Exercise 1.3.1:

Find objections to Artificial Intelligence that have been conveyed in the popular press and in academic papers. What are the fears that lie behind the objections that are put forward as arguments against the development of Artificial Intelligence? Are these objections legitimate, being based upon sound scientific evidence, or are they ill informed? Devise arguments in favour of Artificial Intelligence that address the fears expressed in the objections you have found. Become the devil's advocate and argue the opposite point of view.

Exercise 1.3.2:

Find out arguments for and against Searle's Chinese Room in the literature. Which arguments do you agree with? Do you think that his characterisation of computer-based natural language processing (NLP) as a purely mechanical process that has no ability to understand human language, is valid or too simplistic in light of the many different techniques that are now available for NLP? If you agree with Searle, then how certain are you that there will *never* be (as Searle proclaims) computer programs that can automatically acquire understanding of human language through learning? (For example, through the use of yet-to-be-invented NLP techniques combined with other methods such as situated, embodied agents).

1.4 Conceptual Metaphor, Analogy and Thought Experiments

Exercise 1.4.1:

The second paragraph in Section 1.2 of the companion book "*Artificial Intelligence: Agents and Environment*" uses the following conceptual metaphors: 'paths they can explore', 'unknown terrain', 'paths that are easy going', 'fertile lands', 'paths that lead to mountainous and difficult terrain, or to deserts', 'paths that lead to impassable cliffs'. Find examples of the use of conceptual metaphor throughout the rest of the chapter.

1.5 Design Principles for Autonomous Agents

Exercise 1.5.1:

Read Pfeifer and Scheier's book "*Understanding Intelligence*" to learn more about the design principles they propose. (Reference: Pfeifer, Rolf and Scheier, Christian. 1999. *Understanding Intelligence*. MIT Press.)

2 Agents and Environments

2.1 What is an Agent?

Exercise 2.1.1:

Use your favourite search engine to find academic papers that describe the use of 'agents'. Analyse these papers to determine which of the following perspectives lay behind the meaning they adopt of the term 'agent': an Artificial Intelligence perspective; a distributed computing perspective; an Internet-based computing perspective; a simulation and modeling perspective; or a combination of these perspectives. (See Table 2.1 in the book *Artificial Intelligence – Agents and Environments* for a description of these perspectives).

2.2 Agent-oriented Design Versus Object-oriented Design

Exercise 2.2.1:

Devise an agent-oriented solution and an object-oriented solution for the task of washing a car. Discuss why it is an agent-oriented solution or an object-oriented solution. Also list the objects and the agents for both solutions.

2.3 A Taxonomy of Autonomous Agents

Exercise 2.3.1:

As you did for Exercise 2.1.1, again use your favourite search engine to find academic papers that describe the use of 'proto-agents'. Compare and contrast the meaning that is being adopted in these papers with the meaning of the term 'agent' in the papers you collected for Exercise 2.1.1. How is the meaning different? How is it similar?

Exercise 2.3.2:

Devise your own taxonomy that is different to the one shown in Figure 2.3 of the book *Artificial Intelligence – Agents and Environments* based on the papers you collected for Exercises 2.1.1 and 2.3.1.

2.4 Desirable Properties of Agents

Exercise 2.4.1:

Label the following as a weak agent, a strong agent or not an agent at all. Explain your reasoning for each:

1. Google's web crawler Googlebot.
2. A program set up on a website to collect answers for a questionnaire.
3. A program for a supermarket to automatically locate and bid for the lowest food prices on its Extranet.
4. A distributed information retrieval program that helps you locate Web documents that you are interested in.
5. A mail-filtering program that among other things removes SPAM messages from your email.
6. A multi-user Internet-wide game playing program.
7. A "chatterbot" program whose task is to send messages to chat-rooms and fool the humans into believing it is a real human and not a program.

Exercise 2.4.2:

In the papers you collected for Exercises 2.1.1 and 2.3.1, try to classify the agents and agent-oriented systems mentioned in the papers according to the weak, strong and strongest agent properties. Find agents under the headings listed in the table below for the various industry sectors. Then fill in the table for each agent-oriented system, noting down the URL, the purpose of the site, and indicating if it does or does not have the respective properties (i.e. ticks and crosses may be sufficient; but in most cases, you will need to qualify your answer). The column headings have been deliberately left vague, and you are free to choose anything that is related to the heading.

Property	Health	Government	Banking & Finance	Retail	Media	IT	Unusual / Misc.
Autonomy							
Reactivity							
Proactivity							
Social ability							
etc.							

Which properties present problems when it comes to categorising the various agent-oriented systems? Why?

2.5 What is an Environment?

Exercise 2.5.1:

For the situated agents listed in the left column of the table below, determine which of the attributes listed in the right column (described in Table 2.7 of the book *Artificial Intelligence – Agents and Environments*) are appropriate for describing the environments the agents are situated within.

Agent	Environmental attributes
• A human agent in the real world. • An avatar in Second Life. • The Mars rover robots, Spirit and Opportunity. • CGI generated Uruk-hai in the Lord of the Rings movies. • A chatbot for a Web site. • An NPC (Non-Playing Character) in a First Person Shooter (FPS) video game.	• Observable and partially observable. • Deterministic, stochastic and strategic. • Episodic and sequential. • Static and dynamic. • Discrete and continuous. • Single-agent and multiple-agent.

2.6 Environments as *n*-dimensional spaces

Exercise 2.6.1:

Try out the N Dimensional Space model in NetLogo:

N Dimensional Space	http://files.bookboon.com/ai/N-Dimensional-Space.html

(See also Exercises 3.6.5 and 3.6.6 for further details about the model).

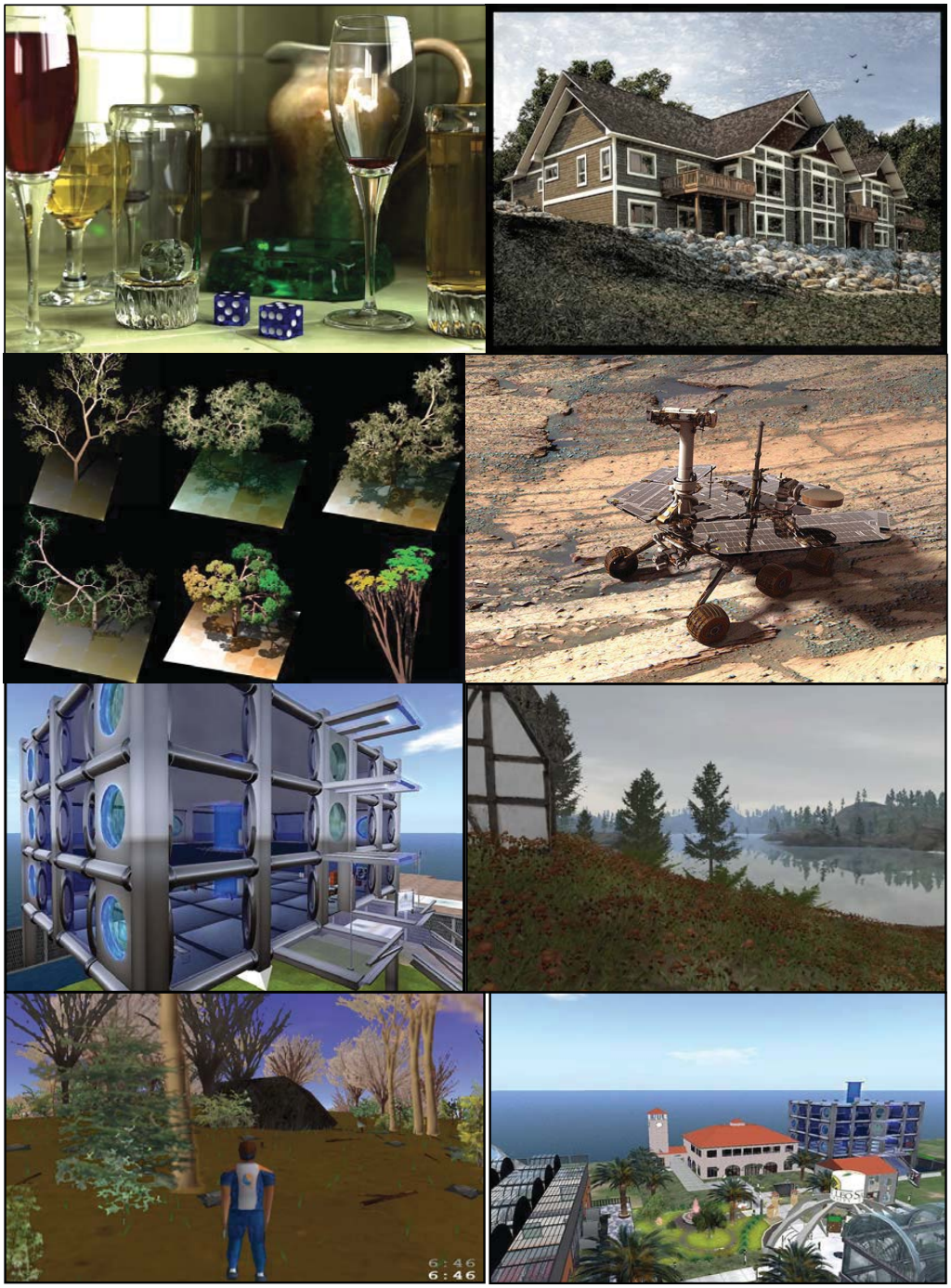
Play with the model by pressing the `setup` button, followed by the `setup-left-plots` and `setup-right-plot` buttons. Compare the two different types of plots – the left plots that use the Cartesian co-ordinate system and the right plot that uses a parallel co-ordinate system. What are the problems with depicting the 5-dimensional data shown by the model for each of the co-ordinate systems?

2.7 Virtual Environments

Exercise 2.7.1:

Classify the following computer-generated images according to the three categories below:

- Pseudo-realistic;
- Photo-realistic;
- Non-photorealistic.



2.8 How can we develop and test an Artificial Intelligence system?

Exercise 2.8.1:

Re-examine the images shown for Exercise 2.7.1. Which of the images depicts environments that could be complex enough for developing and testing an agent that exhibits Artificial Intelligence? Which of the environments provide for the simultaneous involvement of human agents at the same time (who might be able to take on the role of a teacher, for example, to tutor the AI agent as it learns)?

3 Frameworks for Agents and Environments

3.1 Architectures and Frameworks for Agents and Environments

Exercise 3.1.1

Find out how the following computer science terms are being defined in computer science literature and in dictionaries: 'architecture'; 'framework'; 'platform'; 'toolkit' and 'structure'. Are they being consistently defined, or does the definition vary according to the publication? Is the definition for one of the terms in one publication similar to the definition for a different term in another publication?

3.2 Standards for Agent-based Technologies

Exercise 3.2.1

Look up the latest FIPA specifications for agents and multi-agent systems. Find out the categories under which the specifications are listed. Consider how these specifications could be improved to cover further aspects of situated, embodied agents interacting within complex environments.

3.3 Agent-Oriented Programming Languages

Exercise 3.3.1

Look at other agent-oriented programming languages and development environments. Compare and contrast the more popular ones. Why do you think no agent-oriented programming language has yet to become popular the way agent-oriented languages such as C/C++ and Java have become popular?

3.4 Agent Directed Simulation in NetLogo

Exercise 3.4.1

Familiarise yourself with the documentation that comes with the NetLogo programming environment. You will need to refer to this often if you wish to develop your own models. Once you are in NetLogo, select `Help` from the application menu at the top of the screen, then select `NetLogo User Manual`. You can find the following tutorials to learn a bit more about NetLogo:

- Sample Model: Party
- Tutorial #1: Models
- Tutorial #2: Commands
- Tutorial #3: Procedures

For language references, you have the following choices:

- Interface Guide
- Programming Guide
- Transition Guide
- NetLogo Dictionary

The `Interface Guide` provides a summary of the NetLogo user interface and how to navigate around it. Most of the user interface is fairly obvious, so we will not repeat that material here. If you are having difficulties in finding things in the user interface, then this is the place to find about the user menus and user options available.

The `Programming Guide` provides a summary to key elements of the NetLogo programming language. It provides an overview of the language's important features, and can be a useful source for programming examples. It is worth a read, especially when first starting out with the language. The `NetLogo Dictionary` is where NetLogo developers will spend most of their time and it contains links to all the commands that are available in the language. The `Transition Guide` describes earlier versions of NetLogo and what has changed in latter versions, so is therefore of less benefit for someone learning how to program in NetLogo.

Exercise 3.4.2

Perhaps the single most useful thing to know concerning the user interface in NetLogo is how to load the Models Library and select a specific model. In the File Menu, select "`Models Library`" and this will load a large library of models separated into different subject areas such as Art, Biology, Computer Science, Chemistry and Physics and so on. To familiarize yourself with the interface, select the Biology subject area, then select the `Termites` model. After clicking on the `Setup` button, your should have the following appearing on your screen:

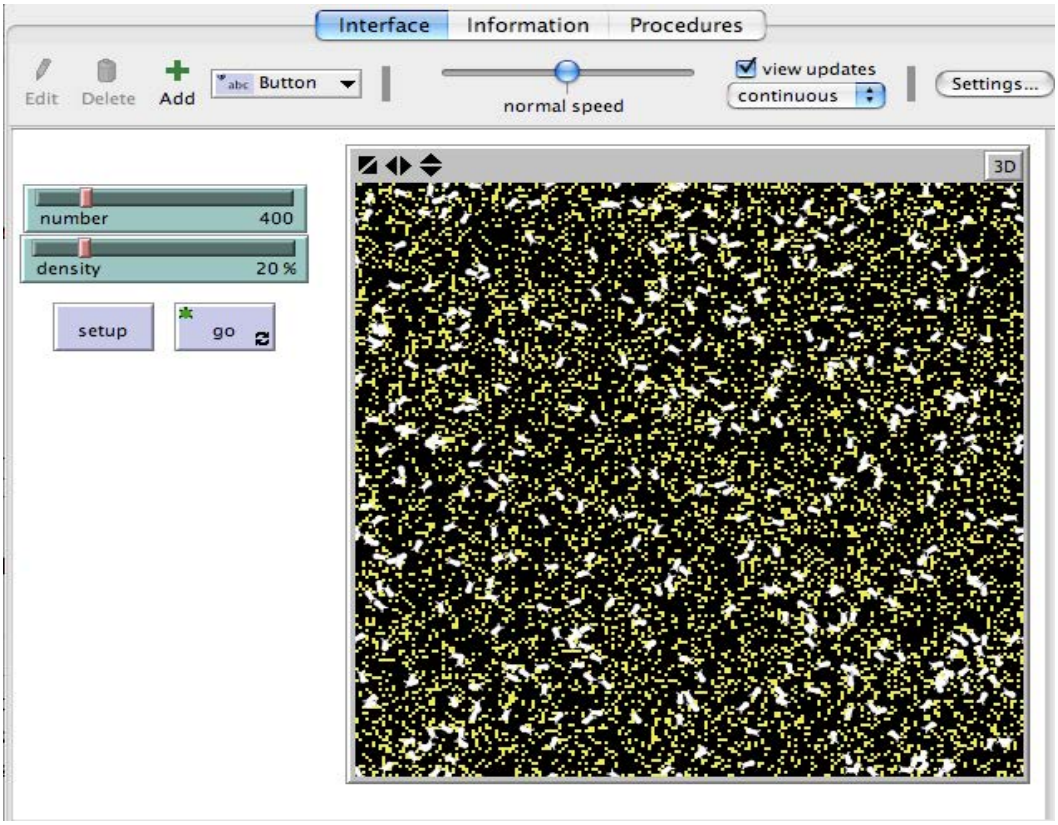
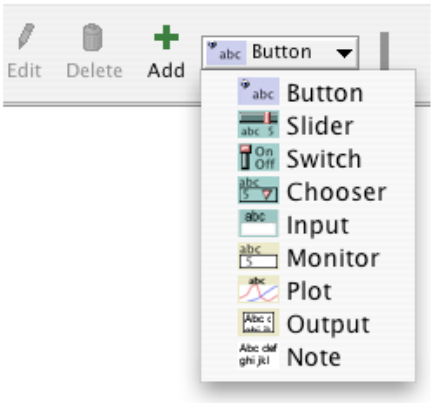


Figure 3.4.2. Screenshot of the Interface for the Termites model after the setup and go buttons have been pressed.

Note the three menu tabs at the top – `Interface`, `Information` and `Procedures`. Select the `Interface` tab. You will be provided with an interface to your program and a visualization of the current state of the (usually 2D) NetLogo environment as shown in Figure 3.4.2. This will vary depending on the model or application currently being executed. Usually there is a `setup` button to set up the initial state of the environment, and a `go` button to start the simulation. Sometimes a `go-once` button might be provided – this will execute the model through a single time step or tick. At other times, a `go-forever` button will be provided which will execute the model indefinitely. These interface elements can be added by clicking on the `Button` menu and selecting from a number of items such as buttons, sliders and choosers:



Various of these interface elements define global variables or require a specific command (i.e. procedure) to be defined in the program somewhere. For example, the `go` button requires a command called `'go'` to be defined somewhere in the model's procedures, but the label displayed on the button can be overridden if required.

Have a go at adding one or more interface elements into the `Interface` for the Termites model.

The `Information` tab switches to a screen where information is displayed about the model that the developer has provided. Find out what `Information` has been included with the Termites model.

The `Procedures` tab will switch to a screen mode where the model's NetLogo code is displayed and can be edited. For the Termites model, find out which code is executed when the setup and go buttons are clicked.

Exercise 3.4.3:

There are four types of agents in NetLogo:

1. turtles;
2. patches;
3. links;
4. the observer.

Explain how the four types of agents are used in NetLogo, using examples from the Models Library.

Exercise 3.4.4:

Try out the Wolf Sheep Predation model in NetLogo:

Wolf Sheep Predation	NetLogo Model's Library: Biology > Wolf Sheep Predation http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

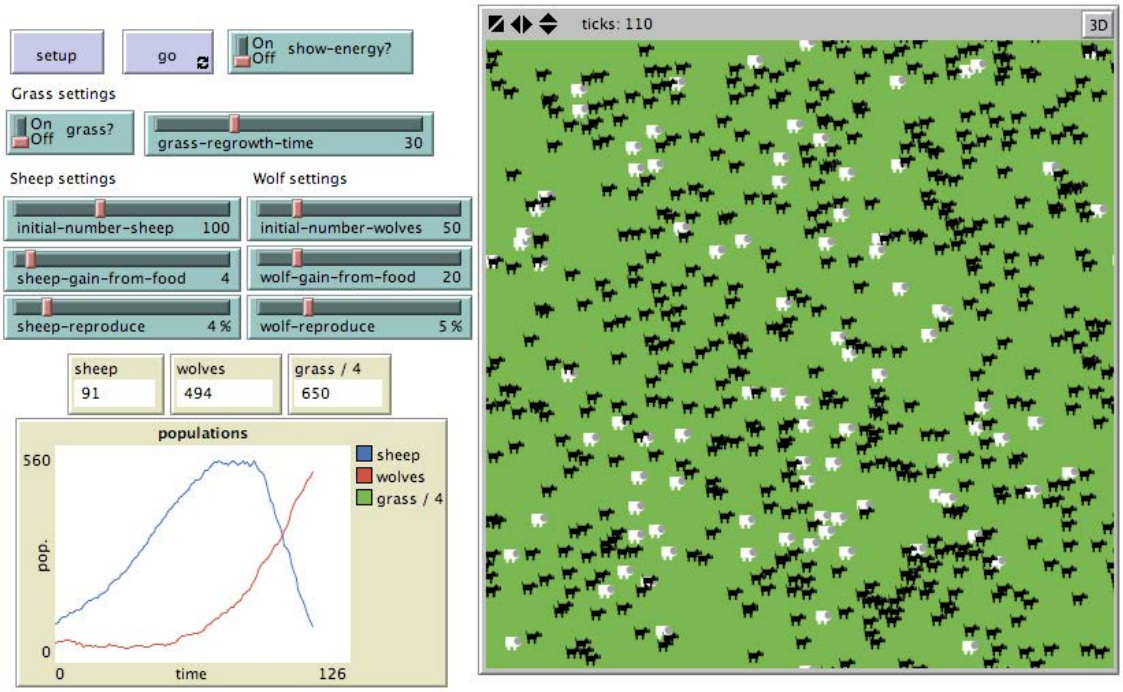


Figure 3.4.4. Screenshot of the Interface for the Wolf Sheep Predation model after the setup and go buttons have been pressed.

Read the `What is it?` and `Things to Notice` sections in the `Information` for the model. Then try out the exercises suggested in the `Things to try` section.

How are the sheep and wolves reproduced in this model? How does a wolf catch a sheep and what happens when it does?

There are multiple ways that sheep die. What are these? And how do the wolves die?

How does the grass grow?

Run the model several times with the default parameters. What are the possible outcomes of the simulation? Try changing the slider values to see what effect they have on the simulation. Can you create different simulation outcomes?

Exercise 3.4.5:

Try out the Tumor model in NetLogo:

Tumor	NetLogo Model's Library: Biology > Tumor http://ccl.northwestern.edu/netlogo/models/Tumor

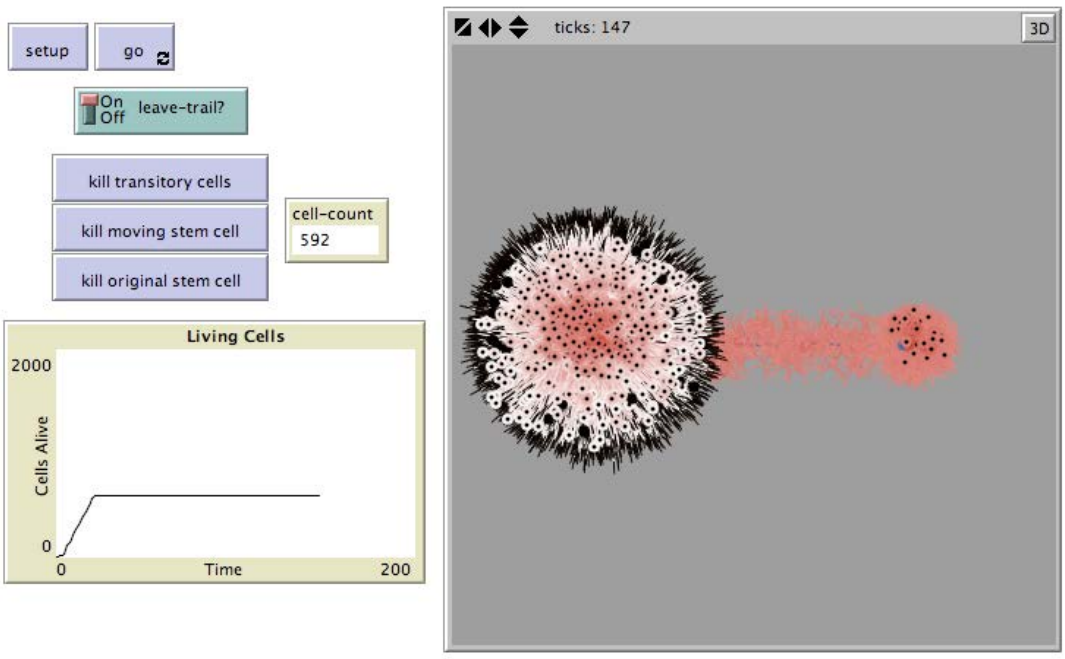


Figure 3.4.5. Screenshot of the Interface for the Tumor model after the setup and go buttons have been pressed.

Read the `What is it?` and `Things to Notice` sections in the `Information` for the model. Then try out the exercises suggested in the `Things to try` section.

How are the following turtle variables used in the model?

- `stem?;`
- `age;`
- `metastatic?.`

What happens when the `mitosis` procedure is called?

Count the ways that different turtle agents can die. Explain what happens for each occurrence.

Exercise 3.4.6:

Discuss the limitations of the agent-oriented approach adopted by NetLogo. What properties do the agents in NetLogo exhibit? For example, do they exhibit any of the weak, strong and strongest properties of agents? (See Section 2.4 of the companion book "*Artificial Intelligence – Agents and Environments*".)

3.5 The NetLogo development environment

Exercise 3.5.1:

Try out the Termites model in NetLogo:

Termites	NetLogo Model's Library: Biology > Termites http://ccl.northwestern.edu/netlogo/models/Termites

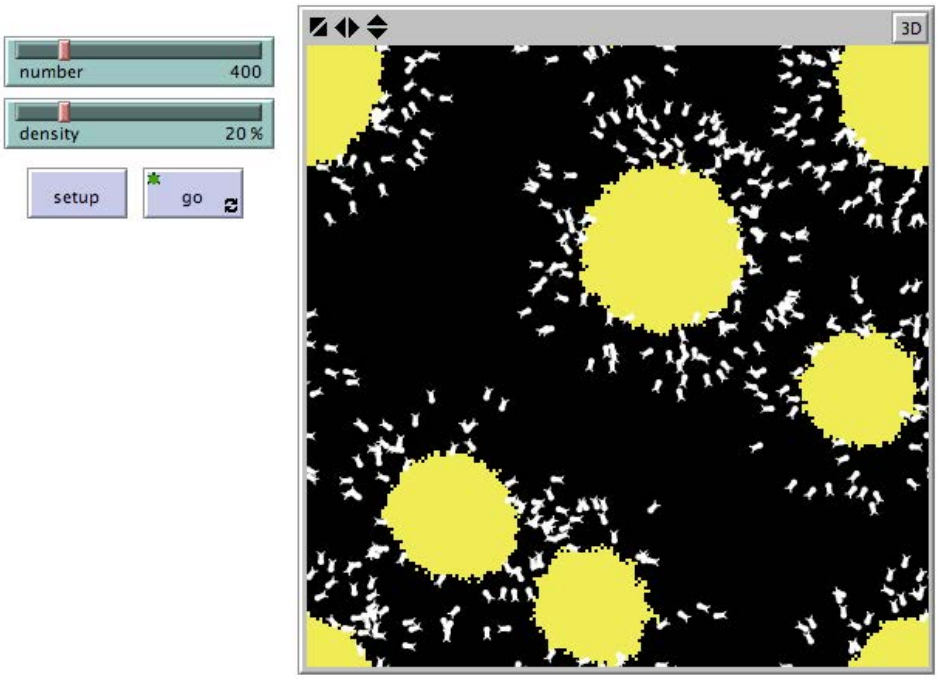


Figure 3.5.1. Screenshot of the Interface for the Termites model after the setup and go buttons have been pressed and the simulation has been running for some time.

Read the `What is it?` and `Things to Notice` sections in the `Information` for the model. Then try out the exercises suggested in the `Things to try` section.

Exercise 3.5.2:

Explain what the following procedures do in the Termites model:

- `search-for-chip;`
- `find-new-pile;`
- `put-down-chip;`
- `get-away;`
- `wiggle.`

Exercise 3.5.3:

Also try out the Termites (Perspective Demo) model in NetLogo:

Termites (Perspective Demo)	NetLogo Model's Library: Perspectives Demo > Termites http://ccl.northwestern.edu/netlogo/models/Termites(PerspectiveDemo)

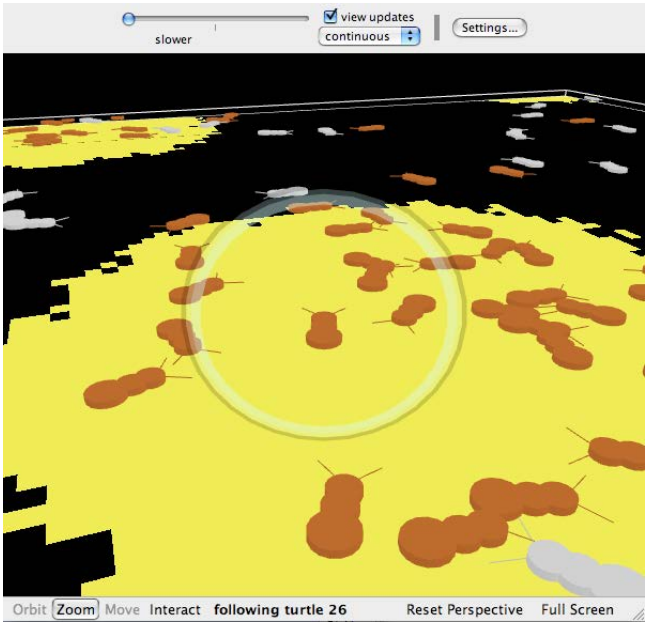


Figure 3.5.3. Screenshot of the 3D View for the Termites (Perspective Demo) model after the setup and go buttons have been pressed followed by the follow one-of turtles button.

To reproduce the screenshot shown in Figure 3.5.3, press the 3D button in the environment once the simulation has started. Then press the `follow one-of turtles` button. Slow down the simulation using the speed slider in the 3D View (using the slider as shown at the top of Figure 3.5.3). This will allow you to discern the individual behaviour of the termite being followed. Verify the behaviour against the source code. Try to determine when each of the three procedures – `search-for-chip`, `find-new-pile`, `put-down-chip` – is being executed as you watch the termite's actions.

Exercise 3.5.4:

Try out the other models in NetLogo's Models Library. Look in the different sub-sections listed under the `Sample Models` section (`Art`, `Biology`, `Chemistry & Physics`, `Computer Science`, `Earth Science`, `Games`, `Mathematics`, `Networks`, `Social Science`, and `System Dynamics`) to learn about the various models that are available.

Especially try scrolling down through the models listed in the `Code Examples` section of the library. A good way of becoming familiar with the models in this section is to press the down arrow to scroll through the models using the menu on the left. These models are often a good starting point for developing your own models.

As an example, have a look at Climate Change model:

Climate Change	NetLogo Model's Library: Earth Science > Climate Change http://ccl.northwestern.edu/netlogo/models/ClimateChange

You can find out about this model by clicking on the `Information` button once it has been loaded.

Click on the `setup` button to set up the environment. Then click `go` to run it. Try adding Clouds and CO_2 to see what happens. Try also changing the `sun-brightness` and `albedo` parameters using the sliders provided in the `Interface`.

Your screen should look like the following:

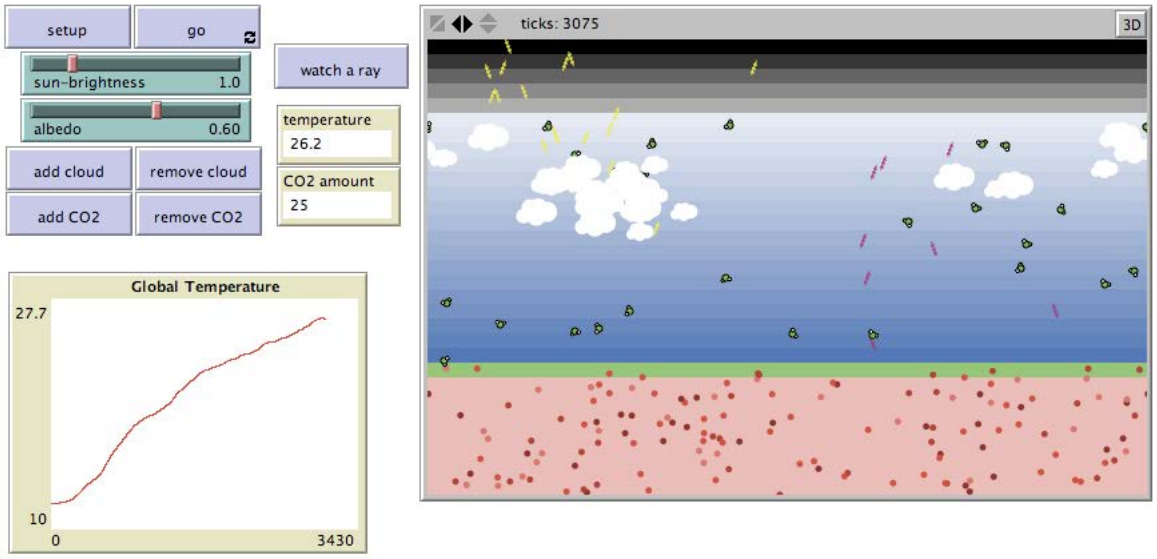


Figure 3.5.4. Screenshot of the Interface for the Climate Change model after the setup and go buttons have been pressed with some clouds and CO_2 added subsequently using the add cloud and add CO2 buttons.

Run the simulation using different settings with different amounts of cloud and CO_2 added. What conditions cause the global temperature to rise?

Exercise 3.5.5:

Try creating your own NetLogo model that will load and display text from a file. Your model will need to consider the three components that make up a model – the `Interface`, the `Information` and the `Procedures`. For the `Interface`, use two buttons, one to load text from a file, and the other to load source code from a NetLogo model file (these are stored on disk using the '.nlogo' extension). Display the text once loaded in a multi-line `Input` box. Add appropriate documentation to the model's `Information`. Make sure you add various sections that detail what the model does, how it does it, how to use it, a description of the user interface, things to notice, things to try and how the model could be extended.

Exercise 3.5.6:

Try exporting the model you created for Exercise 3.5.5 to an HTML text document by using the `Save as Applet` option in the `File` menu. Verify that the applet works when you load the HMTL document into a Web browser.

Exercise 3.5.7:

NetLogo is implemented in Java. We can add new commands as an extension to NetLogo by writing some code in Java. Have a quick look at the `Extensions Guide` in the NetLogo User Manual so that you are aware that this facility is available.

Find out answers to the following questions:

1. How are extensions used in NetLogo?
2. Where are extensions located?

Exercise 3.5.8:

Two particularly useful extensions to NetLogo are the `array` and `table` extensions. You can read more about them by clicking on `Arrays and Tables` in the NetLogo User Manual. Perhaps surprisingly for anyone used to programming in other languages, arrays and tables are seldom used – for example, very few models in the NetLogo Models Library make use of these facilities.

Why do you think this is the case? (Hint: What mechanisms are available in NetLogo that allow you to avoid using arrays and tables?) In what circumstances would you want to use either the array or table extensions?

3.6 Agents and Environments in NetLogo

Exercise 3.6.1:

Describe in detail what the following NetLogo model does:

```
breed [wolves wolf]
breed [sheep a-sheep]
turtles-own [age gender]

to setup
   clear-all

   create-wolves 50 [
     set age 0
     set size 2
     set color brown
     ifelse random 2 = 0
       [set gender "Male"]
       [set gender "Female"]
     setxy random-xcor random-ycor
   ]

   create-sheep 500 [
     set age 0
     set size 2
     set color white
     ifelse random 2 = 0
```

```
            [set gender "Male"]
            [set gender "Female"]
          setxy random-xcor random-ycor
        ]
    end
```

In your answer, identify the turtle agents, the command procedure and the turtle variables. What would happen in the environment when `setup` is called?

Exercise 3.6.2: Foxes and Rabbits NetLogo Model

Try out the Foxes and Rabbits model in NetLogo:

Foxes and Rabbits	http://files.bookboon.com/ai/Foxes-and-Rabbits.html

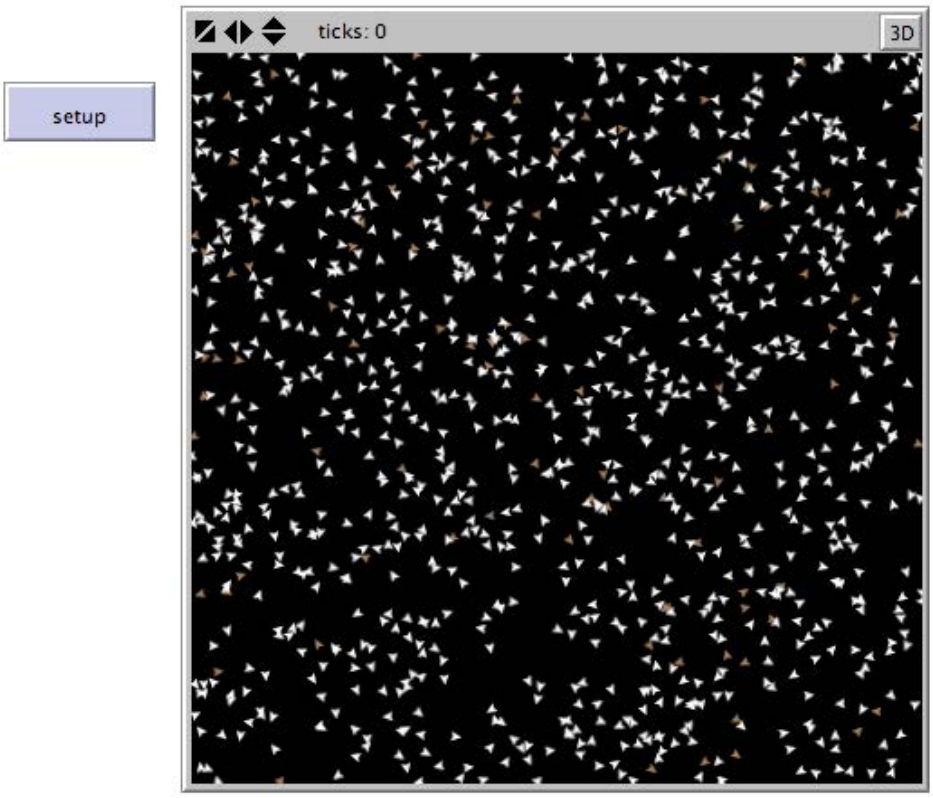


Figure 3.6.2.1. Screenshot of the Interface for the Foxes and Rabbits model after the setup button has been pressed.

WHAT IS IT?

This model shows how to use breeds to create two different breeds of turtle agents – foxes and rabbits – that can have different properties such as colour. It creates 100 fox agents and 1000 rabbit agents at random locations in the environment.

THE INTERFACE

The `setup` button will recreate a new population of foxes and rabbits at random locations.

HOW IT WORKS

It uses two turtle agent breeds, foxes and rabbits, and shows how you can use the `turtles-own` command so that both breeds end up with common variables – `age` and `gender`.

The `setup` command simply calls the `create-foxes` and `create-rabbits` command to create the agents.

HOW TO USE IT

You can't really use it for anything, except for pressing the `setup` button several times to see how the agents spread themselves throughout the environment.

WHAT IS ITS PURPOSE?

Its purpose is to show how to define and create breeds of agents.

EXTENDING THE MODEL

Try adding two sliders that control the number of agents created when the `setup` button is pressed.

Try changing the shapes of the agents so that they look like foxes and rabbits. (Hint: You will need to use the Turtle Shapes Editor to create the shapes.) Try also changing the sizes of the agents so that foxes are bigger then the rabbits. Your model should look like that shown in Figure 3.6.2.2.

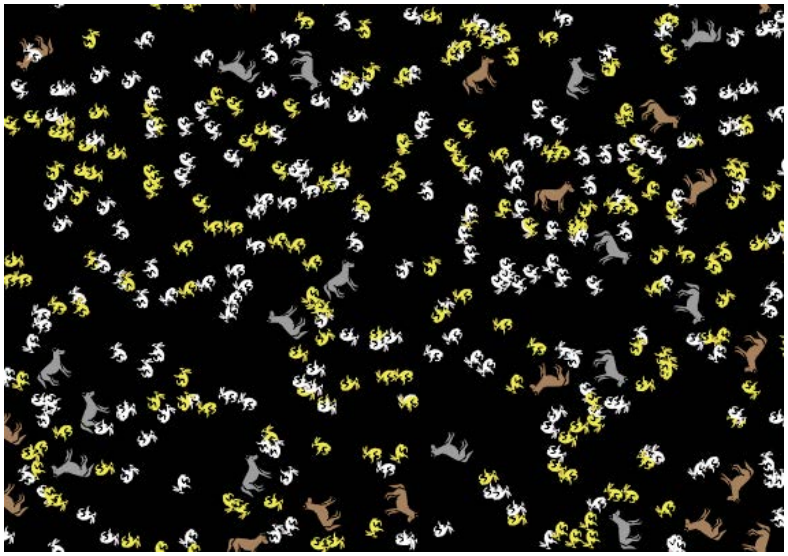


Figure 3.6.2.2. Screenshot from the Foxes and Rabbits 2 model.

RELATED MODELS

See the Foxes and Rabbits 2 model.

Exercise 3.6.3: Life Example NetLogo Model

As a further example of how to define your own breeds of turtle agents in NetLogo, try out the following Life Example model in NetLogo. Also look at the code by pressing the `Procedures` tab in the `Interface` and try to figure out what it is doing.

Life Example	http://files.bookboon.com/ai/Life-Example.html

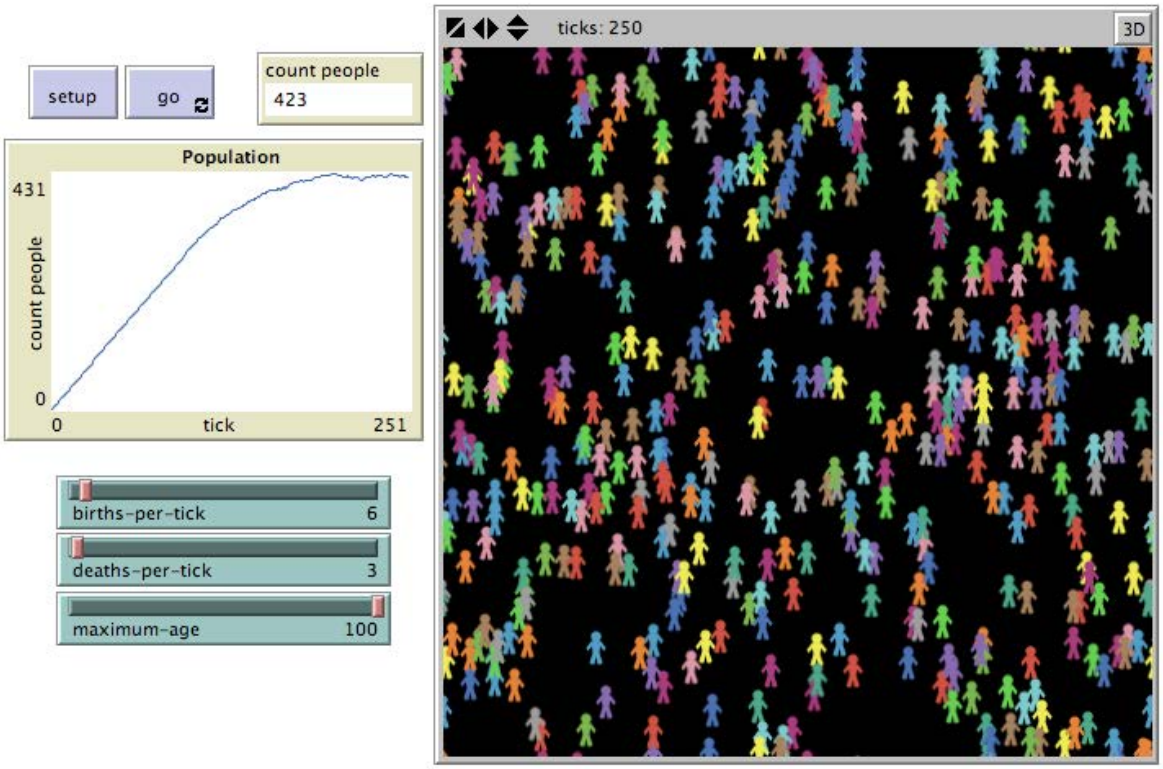


Figure 3.6.3.1. Screenshot of the Interface for the Life Example model after the setup and go buttons have been pressed, with the slider values as shown.

WHAT IS IT?

This model shows how to use some simple commands in NetLogo to simulate the life cycle of people.

THE INTERFACE

The `setup` button will reset the simulation to the start state when nobody is alive.

The `go` button will run the simulation.

The `births-per-tick` slider controls the number of births per tick of the simulation.

The `deaths-per-tick` slider controls the number of deaths per tick of the simulation.

The `maximum-age` slider specifies the maximum age before people die of old age.

The `"count people"` monitor reports the number of people currently alive.

The `Population` plot graphs the number of people versus tick.

HOW IT WORKS

It uses one turtle agent breed, `people`, that has a single variable – `age`. Births and deaths occur every time tick as according to the `births` and `deaths` procedures. The `people` turtle agents live until they are chosen randomly to die early, or until they grow too old, as according to the `life` procedure.

HOW TO USE IT

Select how many births and deaths you want per tick, and the maximum age, using the three sliders. Press the `setup` button to reset the simulation. Then press the `go` button to start it.

WHAT IS ITS PURPOSE?

Its purpose is to show how to define and create breeds of agents.

THINGS TO NOTICE

Notice what happens when you set the number of deaths greater than or equal to the number of births. If you do this at the start, nothing will happen. Why?

If you set the number of births to one more than the deaths at the beginning temporarily, with the maximum age set at 100, then not long afterwards while the simulation is still running try increasing the number of deaths so that it equals the number of births. Notice that the population gradually reduces until eventually it will reach an equilibrium point. Notice that it takes a long time for this to happen. Why is this?

Notice that when the number of births is set to low, it seems that the people are moving around the environment. However, this is just an illusion. There is no code in the model that makes the people turtle agents move around.

THINGS TO TRY

Try changing the values of the sliders to see what happens in the simulation.

Try changing these values as the simulation is running and observe what happens in the plot.

EXTENDING THE MODEL

Try adding further parameters to the model to make the simulation more and more like real-life.

RELATED MODELS

Compare this model with the Hatch Example model that comes with NetLogo's Models Library:

Hatch Example	NetLogo Model's Library: Code Examples > Hatch Example http://ccl.northwestern.edu/netlogo/models/HatchExample

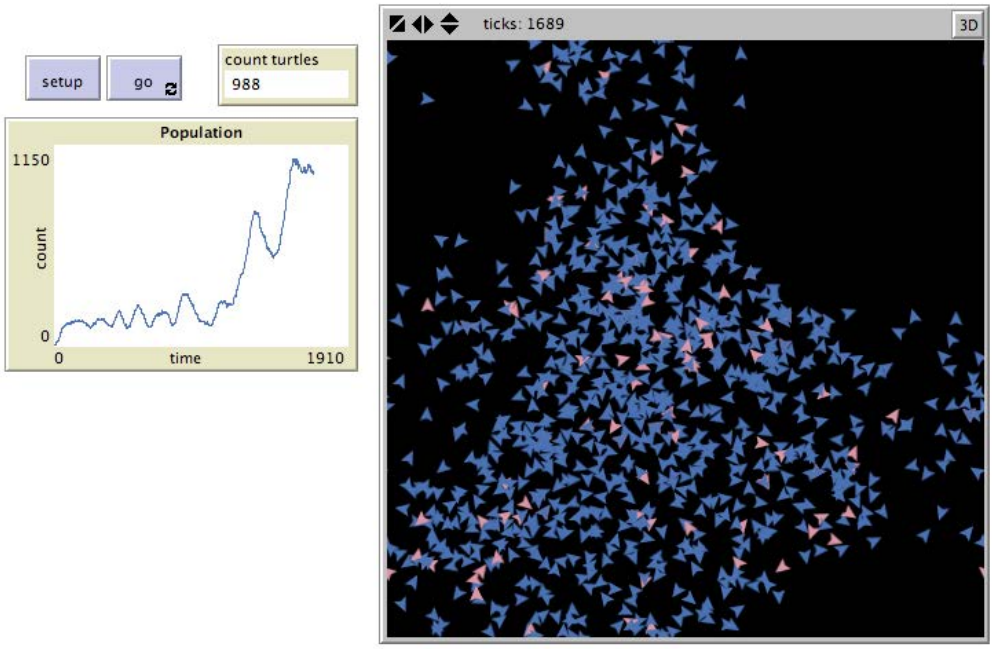


Figure 3.6.3.2. Screenshot of the Interface for the Hatch Example model after the setup and go buttons have been pressed, with the slider value as shown.

Notice with this model that the sustainability of the population is very sensitive to the initial conditions and can also oscillate between growing and slowly dying out. Often, the entire population will die out completely, but occasionally, the population will survive for longer and begin to thrive. Under what conditions does this occur? In comparison, under what conditions does the population for the Life Example model thrive or die out?

Exercise 3.6.4:

Explain what the following procedures do in the Life Example model:

- `births;`
- `life;`
- `deaths.`

Exercise 3.6.5: N Dimensional Space NetLogo Model

Try out the N Dimensional Space model in NetLogo:

N Dimensional Space	http://files.bookboon.com/ai/N-Dimensional-Space.html

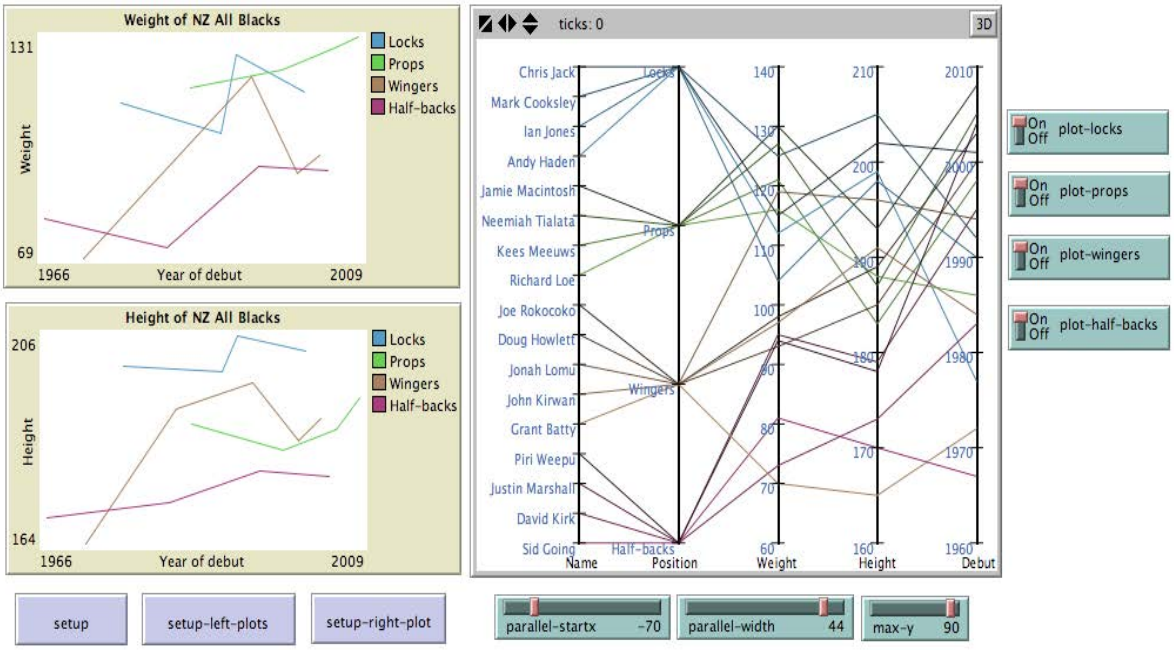


Figure 3.6.5. Screenshot of the Interface for the N Dimensional Space model after the setup, setup-left-plots and the setup-right-plot buttons have been pressed.

WHAT IS IT?

This model plots a simple set of 5-dimensional data (about 17 New Zealand All Blacks rugby players) using Cartesian co-ordinates as well as parallel co-ordinates.

THE INTERFACE

The three buttons at the bottom left are defined as follows:

- `setup`: clears both plots;
- `setup-left-plots`: plots the data using Cartesian co-ordinates in the left hand plots;
- `setup-right-plot`: plots the data using parallel co-ordinates in the right hand plot.

The three sliders in the middle bottom are defined as follows:

- `parallel-startx`: where the leftmost vertical axis is drawn for the right hand plot;
- `parallel-width`: the width between vertical axes for the right hand plot;
- `max-y`: the maximum height from $y = 0$ of each vertical axis (highest y co-ordinate is (+ `max-y`) and lowest y co-ordinate is (– `max-y`).

The five switches to the right are defined as follows when set to `On`:

- `plot-locks`: this will result in data relating to the locks being plotted;
- `plot-props`: this will result in data relating to the props being plotted;
- `plot-wingers`: this will result in data relating to the wingers being plotted;
- `plot-half-backs`: this will result in data relating to the half-backs being plotted.

HOW IT WORKS

The left plots use the standard NetLogo plotting features to plot the data using the Cartesian co-ordinate system. With 5-dimensional data, a problem occurs when trying to depict the data on a 2-dimensional plot. We can add a third dimension by plotting several lines on the same graph, but that leaves the problem of how to plot the other two dimensions. One solution is to have two graphs, and plot a subset of the dimensions. In the model, the two dimension subsets are: (Weight, Year-of-debut, Position), to show trends in the weights of NZ All Blacks; and (Height, Year-of-debut, Position), to show trends in the heights of NZ All Blacks.

The right plot plots the 5-dimensional data using parallel co-ordinates. It uses patch agents to plot the parallel axes, and labelled points on the axes using the turtle breed axis-points. Each polyline that represents a 5-dimensional point is plotted by having each `all-blacks` turtle agent move from left to right drawing lines between the parallel axes.

HOW TO USE IT

You can use these plots to spot trends or patterns in the data (such as locks and props being generally heavier and taller than wingers and half-backs, except for Jonah Lomu).

WHAT IS ITS PURPOSE?

Its purpose is to show the problem with visualising n-dimensional data, and the difference between plotting using the Cartesian co-ordinate system and the parallel co-ordinate system.

THINGS TO TRY

See what happens in the right plot when you change the value of the sliders. In some cases, the entire plot will no longer fit in the environment. In this case, you will need to alter the `Settings` of the environment to suit.

Try turning off the plotting of the locks, props, wingers or half-backs.

EXTENDING THE MODEL

Try adding the capability where the user has more control over what gets plotted. For example: allow the user to decide on which 3-dimensional subset of the data is plotted in the left plots; or allow the user to decide on the order the axes are plotted in the right plot.

RELATED MODELS

See the Missionaries and Cannibals model to see parallel co-ordinates used to plot search space states.

Exercise 3.6.6:

In NetLogo, it is very easy to create plots similar to the ones shown on the left in Figure 3.6.5. First of all, you need to create a plot in the `Interface`. You do this by clicking on the `Button` menu, then selecting '`Plot`' as the type of object to add to the `Interface`. When you do this, you will be able to place a new plot inside the `Interface`. At the same time, you will be asked to give it a name, and specify its attributes. With the name you give (e.g. `"Height of NZ All Blacks"`), you will be able to update the plot inside the code using various commands, such as:

```
set-current-plot "Height of NZ All Blacks"
set-current-plot-pen "Half-backs"
```

By default, a plot has only one pen associated with it. A pen is used to draw the line that is being plotted. You can, however, create multiple pens – in the figures to the left of Figure 3.6.5, there were four pens used as there are four lines plotted (blue, green, brown and magenta). You create an additional pen by clicking on the plot once it has been placed in the `Interface`, then selecting `Edit` to edit the plot, and then clicking on `Create` to create a new plot pen. You will then be asked to enter the name for the plot pen – you are free to type in any string; for example, in Figure 3.6.5, the four plot pens were named '`Locks`', '`Props`', '`Wingers`' and '`Half-backs`'.

There are two commands for specifying what is plotted:

```
plot number
plotxy x, y
```

The `plot` command will plot a point at *x*, *y* co-ordinates where *y* = `number` (the number that is passed as the single parameter to the command) and *x* increments from *x* = 0 the first time it is called, then *x* = 1 the second time, *x* = 3 the third time and so on.

The `plotxy` command will plot a point at the specified *x*, *y* co-ordinates that are passed as parameters to the command.

As an exercise, we can look at a small set of data such as that listed in Table 3.6.6. The table lists a set of tuples containing information about New Zealand rugby players, called All Blacks. The data is 5-dimensional as each tuple (a row in the table) contains five attributes about each player – the position they played, their name, their height in cm, their weight in kg, and the year that they first played for the All Blacks.

Position	Name	Height (cm)	Weight (kg)	Debut year
Wing	Grant Batty	165	70	1972
Wing	Doug Howlett	185	93	2000
Wing	John Kirwan	191	97	1984
Wing	Jonah Lomu	196	119	1994
Wing	Joe Rokocoko	189	98	2003
Half-back	Sid Going	170	81	1967
Half-back	David Kirk	173	73	1983
Half-back	Justin Marshall	179	95	1995
Half-back	Piri Weepu	178	94	2004
Prop	Richard Loe	188	116	1986
Prop	Jamie Mackintosh	193	130	2008
Prop	Kees Meeuws	183	121	1998
Prop	Neemiah Tialata	187	127	2005
Lock	Mark Cooksley	205	125	1992
Lock	Andy Haden	199	112	1977
Lock	Chris Jack	202	115	2001
Lock	Ian Jones	198	104	1990

Table 3.6.6 An example of 5-dimensional data: Some New Zealand All Blacks rugby players.

Write a model in NetLogo. Create data structures for storing the data in Table 3.6.6 using the following four methods:

- Lists;
- Arrays;
- Tables;
- Turtle agents.

For the last one, you will need to define a breed for storing the data inside a turtle agent. Create plots using the data that is stored in the four different data structures.

Exercise 3.6.7: Empty Maze NetLogo Model

Try out the Empty Maze model in NetLogo:

Empty Maze	http://files.bookboon.com/ai/Empty-Maze.html

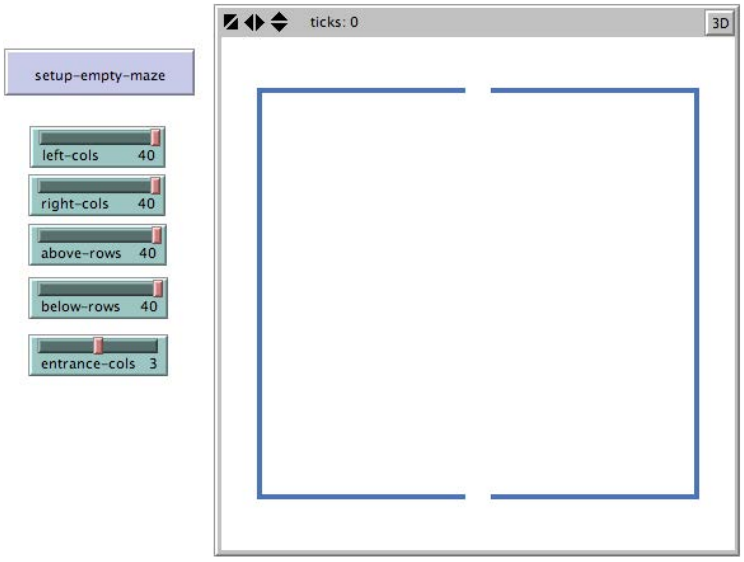


Figure 3.6.7. Screenshot of the Interface for the Empty Maze model after the setup-empty-maze button has been pressed. The entrance to the maze is in the middle bottom, and the goal is to get to the other side of the maze.

WHAT IS IT?

This model sets up an empty "maze" which is a maze with only external walls, and with an entrance and exit at the middle bottom and middle top.

THE INTERFACE

The `setup-empty-maze` button will redraw the maze.

The sliders are defined as follows:

- `left-cols`: width of the horizontal walls drawn to the left of the entrance/exit;
- `right-cols`: width of the horizontal walls drawn to the right of the entrance/exit;
- `above-rows`: height of the vertical walls drawn above the row at $y = 0$;
- `below-rows`: height of the vertical walls drawn below the row at $y = 0$.

HOW IT WORKS

It uses one `ask patches` command to set the patches blue that define the walls, and to set the remaining patches white.

HOW TO USE IT

You can't really use it for anything, except for changing the values in the sliders to create different sized mazes. To have the model draw the maze, press the `setup-empty-maze` button.

WHAT IS ITS PURPOSE?

Its purpose is to show how easy it is to use patch commands to define the environment.

THINGS TO TRY

See what happens when you change the value of the sliders.

Try changing the `Settings` of the environment such as the `Patch size` and the maximum and minimum x and y co-ordinates.

EXTENDING THE MODEL

Try changing the maze so that the entrance and exit are elsewhere.

Try adding a turtle agent to move around the maze.

RELATED MODELS

See the Hampton Court Maze and Chevening House Maze models.

Exercise 3.6.8:

Draw a set of six nested squares as shown in Figure 3.6.8 on the right. However, draw each square in a different way, as follows:

1. using a single turtle agent for the innermost square with the shape `"square 2"`;
2. using a single turtle agent for the next square but drawing it with the pen and the `setxy` command;
3. using a single turtle agent for the next square drawn with the `stamp` command;
4. using multiple turtle agents for the next square;
5. using four link agents between four turtle agents for the next square; and finally
6. using patch agents for the outermost square.

Hint: Look at the Box Drawing Example in NetLogo's Models Library as a starting point.

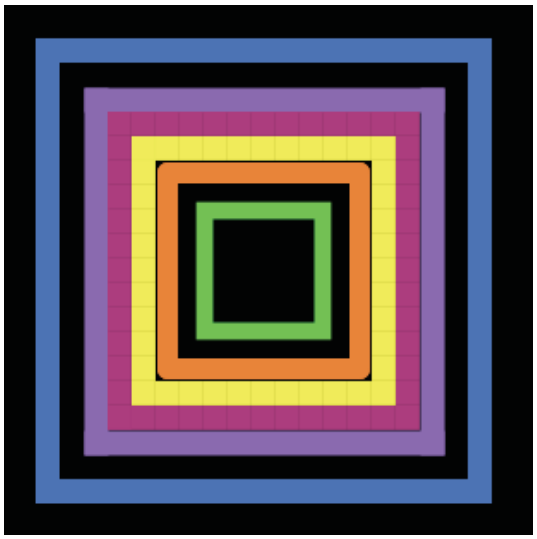


Figure 3.6.8. Six nested squares. Screenshot from the Nested Squares model.

Exercise 3.6.9:

Import your own environments using the `import-pcolors` command. Turtles will be able to sense this environment when they move around within it. Overlay this environment with the original image using the `import-drawing` command. Us, as observers, can now "see" what the environment should look like, but the turtles cannot "see" the full detail – they only can sense the imported patches. Add some turtle agents to your environments (with and without the imported patches via the `import-pcolors` command) to verify which of the commands allow the turtle agents to sense them.

Exercise 3.6.10:

Write a NetLogo model that shuffles and deals a pack of cards. You should use the Turtle Shapes Editor in the Tools menu to create turtle shapes for your pack of cards. You can also use the NetLogo `shuffle` command to shuffle the cards.

Exercise 3.6.11:

Write a NetLogo model that allows the user to fill in the squares of a Sudoku puzzle. It should check that the number being entered is correct – that is, the number does not occur elsewhere horizontally, vertically or within the same 3 by 3 grid.

Exercise 3.6.12:

Write a NetLogo model that reads in data from the following file:

Santa Fe Ant Trail Data File	http://files.bookboon.com/ai/Santa-Fe-Trail.dat

This data file contains data that specifies the Santa Fe ant trail devised by John Koza in order to test the performance of evolutionary algorithms. Your ant trail should look similar to the image as shown in Figure 3.6.12.

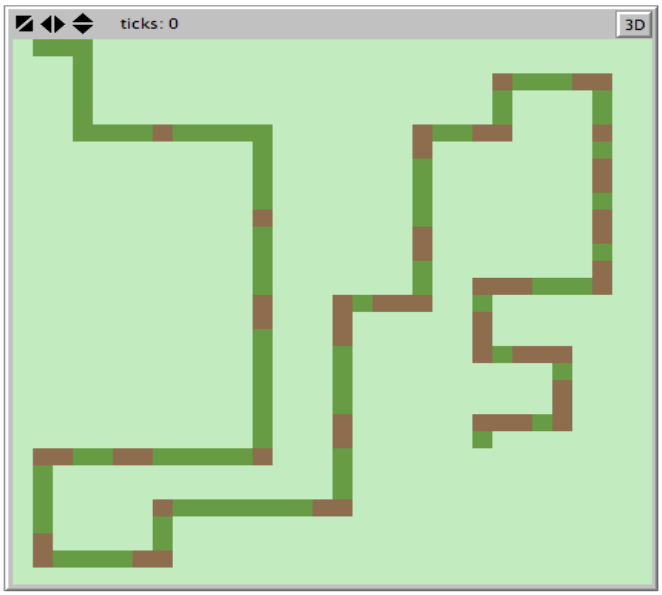


Figure 3.6.12. The Santa Fe Ant Trail.

The data file contains a '`.`' character to indicate that the patch is not on the trail, the '`1`' character to indicate where the food is on the trail, and the '`0`' character to indicate where the trail is but with no food. In the image shown, the non-trail patches are light green, the food patches on the trail dark green, and the non-food patches on the trail are brown.

The Santa Fe Ant Trail problem is to evolve a computer program that will provide the instructions for an ant to follow the trail. This is an exercise that will be investigated in Volume 2 of this book series. However, for the exercise here, all that you need to do is have the model read the trail from the data file and then display it in the environment.

3.7 Drawing Mazes using Patch Agents in NetLogo

Exercise 3.7.1: Hampton Court Maze NetLogo Model

Try out the Hampton Court Maze model in NetLogo:

Hampton Court Maze	http://files.bookboon.com/ai/Hampton-Court-Maze.html

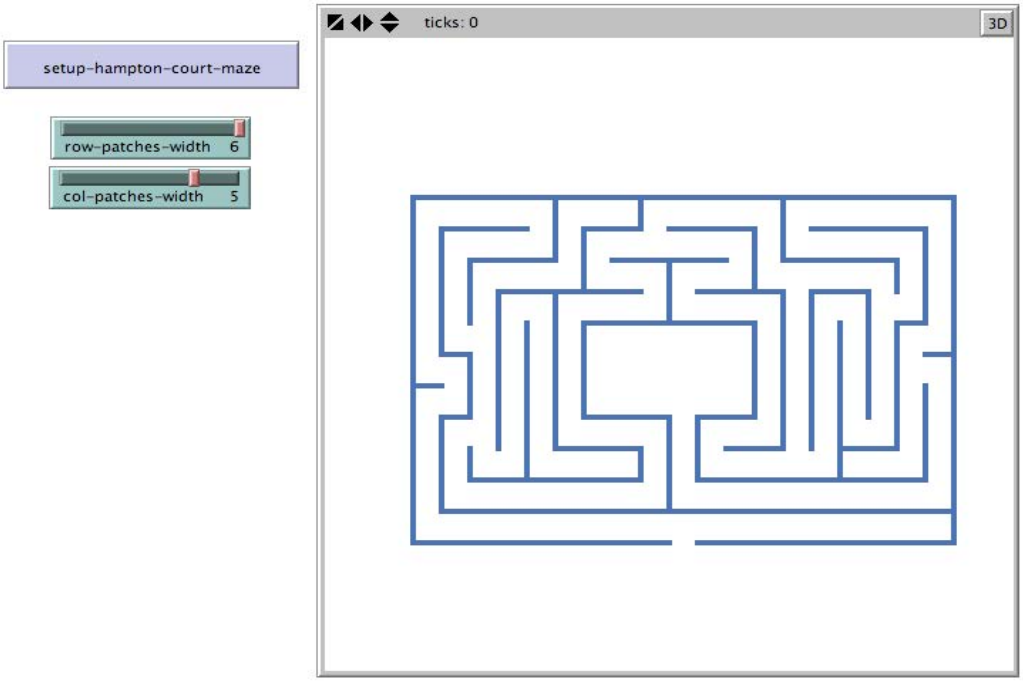


Figure 3.7.1. Screenshot of the Interface for the Hampton Court Maze model after the `setup-hampton-court-maze` button has been pressed. The entrance to the maze is in the middle bottom, and the goal is to get to the centre of the maze.

WHAT IS IT?

This model draws a schematic of the Hampton Court Palace garden maze in the United Kingdom. The entrance is at the middle bottom of the maze, and the goal is to get to the centre.

THE INTERFACE

The `setup-hampton-court-maze` button will redraw the maze.

The sliders are defined as follows:

- `row-patches-width` : the width between horizontal walls in the maze;
- `col-patches-width` : the width between vertical walls in the maze.

HOW IT WORKS

It uses one `ask patches` command to set the patches blue that define the walls, and to set the remaining patches white. The code uses two procedures that are similar, `setup-row` and `setup-col`, to draw the walls in a horizontal row or vertical column respectively. These procedures take the row or column number as the first parameter, the colour that the walls are to be drawn with, and a list containing two-numbered range lists that define the segments where the walls are to be drawn in the row or column.

HOW TO USE IT

You can't really use it for anything, except for changing the values in the sliders to create different sized mazes. To have the model draw the maze, press the `setup-hampton-court-maze` button.

WHAT IS ITS PURPOSE?

Its purpose is to create a virtual maze that has a corresponding maze in real-life to demonstrate how virtual environments can mirror real-life environments, but the reflection can often be distorted in the process.

THINGS TO TRY

See what happens when you change the value of the sliders.

Try changing the `Settings` of the environment such as the `Patch size` and the maximum and minimum x and y co-ordinates.

EXTENDING THE MODEL

Try adding a turtle agent to move around the maze.

RELATED MODELS

See the Empty Maze and Chevening House Maze models.

Exercise 3.7.2: Chevening House Maze NetLogo Model

Try out the Chevening House Maze model in NetLogo:

Chevening House Maze	http://files.bookboon.com/ai/Chevening-House-Maze.html

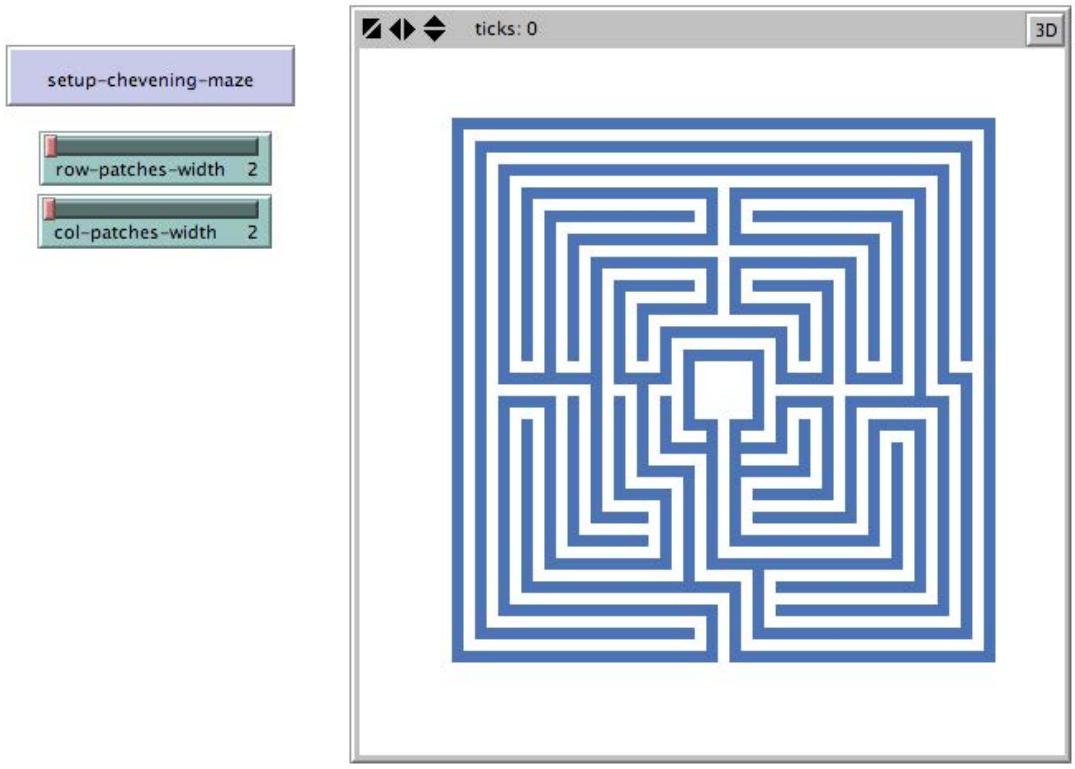


Figure 3.7.2. Screenshot of the Interface for the Chevening House Maze model after the setup-chevening-maze button has been pressed. The entrance to the maze is in the middle bottom, and the goal is to get to the centre of the maze.

WHAT IS IT?

This model draws a schematic of the Chevening House maze in the United Kingdom. The entrance is at the middle bottom of the maze, and the goal is to get to the centre. The maze has been designed to specifically thwart the hand-on-the-wall behaviour for reaching the centre. In this case, the centre occurs on an island which is not directly connected to the outside walls.

THE INTERFACE

The `setup-chevening-maze` button will redraw the maze.

The sliders are defined as follows:

- `row-patches-width`: the width between horizontal walls in the maze;
- `col-patches-width`: the width between vertical walls in the maze.

HOW IT WORKS

It uses one `ask patches` command to set the patches blue that define the walls, and to set the remaining patches white. The code uses two procedures that are similar, `setup-row` and `setup-col`, to draw the walls in a horizontal row or vertical column respectively. These procedures take the row or column number as the first parameter, the colour that the walls are to be drawn with, and a list containing two-numbered range lists that define the segments where the walls are to be drawn in the row or column.

HOW TO USE IT

You can't really use it for anything, except for changing the values in the sliders to create different sized mazes. To have the model draw the maze, press the `setup-chevening-maze` button.

WHAT IS ITS PURPOSE?

Its purpose is to create a virtual maze that has a corresponding maze in real-life to demonstrate how virtual environments can mirror real-life environments but where the reflection can often be distorted in the process.

THINGS TO TRY

See what happens when you change the value of the sliders.

Try changing the `Settings` of the environment such as the `Patch size` and the maximum and minimum x and y co-ordinates.

Try drawing the middle island in another colour to show that it is indeed an island.

EXTENDING THE MODEL

Try adding a turtle agent to move around the maze.

RELATED MODELS

See the Empty Maze and Hampton Court Maze models.

Exercise 3.7.3:

Try creating your own maze using the `setup-row` and `setup-col` procedures that were used in the Hampton Court Maze and Chevening House Maze models to create the two mazes. Try to make it different to the two mazes, with its own unique flavour. For example, try using different colours for the walls, and make it in a shape of an animal or insect, such as a butterfly.

Exercise 3.7.4:

Try modifying the Chevening House Maze model to show why the centre of the maze is not connected to the outside walls. You can do this by using a different colour for each separate "island" in the maze. How many islands are there in the maze?

4 Movement

4.1 Movement and Motion

Exercise 4.1.1:

Try adding your own examples of Distance, Containment or Movement based metaphors to Table 4.1 from Chapter 4 of the book "*Artificial Intelligence – Agents and Environments*" that accompanies these Exercises.

Exercise 4.1.2:

"In standard American English, the word with the most gradations of meaning is probably run. *The Random House Unabridged Dictionary offers one hundred and eight options, beginning with 'to go quickly by moving the legs more rapidly than at a walk' and ending with 'melted or liquefied'."*

Stephen King, Wolves of the Calla, 2003 (page 523).

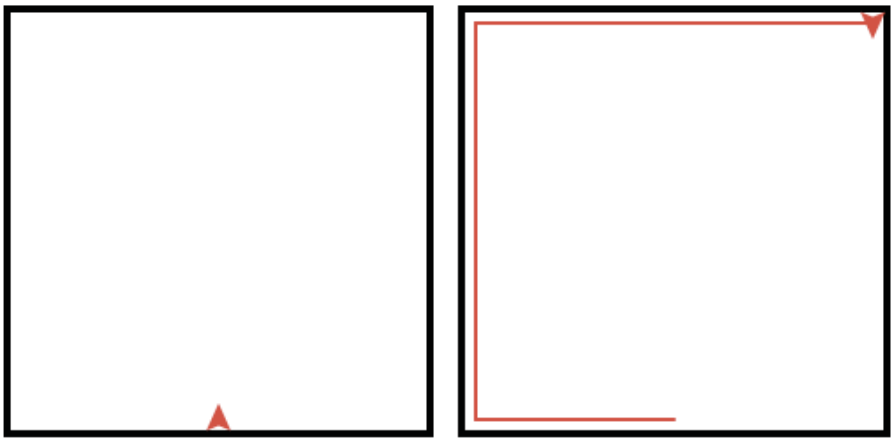
Check this quote by looking at the entry for the word 'run' in several dictionaries. Also check the entries for other movement related words such as 'walk', 'crawl', 'move' and 'go'.

The many meanings for the gradations of the word 'run' spring from the common notion of some form of quicker movement, whether metaphorically or by more direct correspondence with the physical activity of running. Why do you think 'run' has many more gradations in the dictionary entries than for the other movement related words 'walk' and 'crawl'? Note that the words 'move' and 'go' also have many gradations, but not as many as 'run'.

4.2 Movement of Turtle Agents in NetLogo

Exercise 4.2.1:

Write down the set of instructions (using the commands `forward`, `left` and `right`) that will get a turtle agent to navigate from the following starting point shown in the left image below to the point shown in the right image below:



The dimensions of the black square are 60 patches by 60 patches. The turtle starts in the middle bottom of the square as shown in the left image.

Exercise 4.2.2:

Write down the set of instructions for getting the turtle agent to move using a *minimum number of steps* from the starting point in the left image in Exercise 4.2.1 above to the finishing point in the right image.

Exercise 4.2.3:

Write a NetLogo model that uses a turtle agent to draw a set of nested equilateral triangles as shown by the image on the right. The triangles should be drawn with the outer triangle first, followed by the next one inside it, and so on. Add sliders to the `Interface` of your model so that you can change the following: the initial orientation; the initial co-ordinates of the bottom tip of the outer triangle; the number of triangles drawn; the thickness of the sides; the initial length of one of the sides; the jump angle and distance; and the ratio for reducing the length of the sides, when the turtle jumps from drawing one triangle to the next one nested inside it.

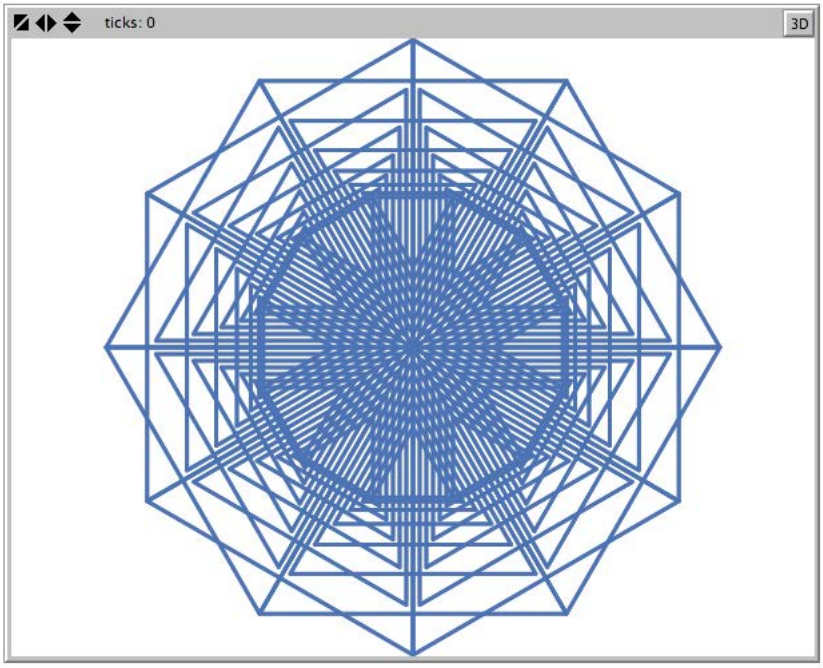
Play around with your model by changing the value of the sliders to see the variety of patterns you can generate. Alter the code so that the colour used to draw each triangle changes each iteration.

Exercise 4.2.4:

Modify your code so that the heading of the turtle can be changed by an angle specified by a slider in the `Interface` once a triangle has been drawn and the turtle has jumped over to the starting point of the next triangle. This will cause the triangles to be rotated incrementally each iteration by the angle specified by the slider. Now have a go at reproducing the image on the right.

Exercise 4.2.5:

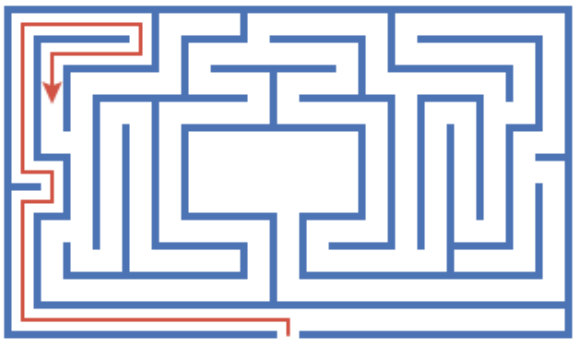
Modify your model so that you can repeat the drawing of the set of nested triangles multiple times. Add a slider to the `Interface` that allows you to control the number of times the set of triangles are drawn, and the angle or rotation that is applied before moving on to drawing the next set of triangles. Have a go at producing the pattern shown in the image below. This was produced by the Nested Triangles NetLogo model which is discussed in the solutions to this exercise at the end of this book.



Play around with the sliders and create two or more distinctive patterns of your own that are different to the ones above. In at least one of your patterns, experiment with changing the colour to create a more aesthetically pleasing pattern.

Exercise 4.2.6:

Add a turtle to the Hampton Court maze and have it move around the maze, as shown in the following image:



Exercise 4.2.7:

Modify the Foxes and Rabbits model from Exercise 3.6.2 so that once the rabbits and foxes have been created, the rabbits move away from the nearest foxes until they feel they are at a safe distance. Add sliders to the `Interface` so that you can control the number of foxes, the number or rabbits and the radius at which the rabbit feels safe.

Write two versions of the rabbits' behaviour where in the first one, all rabbits all move away at the same time, and in the second one, the rabbits take turns to move to a safe distance.

4.3 Behaviour and Decision-making in terms of movement

Exercise 4.3.1:

Add a turtle agent to the model you developed for the Santa Fe Ant Trail in Exercise 3.6.12. (You can use the "bug" shape in the Turtle Shapes Editor to make it look like an ant). Try to get the turtle agent to follow the trail using just three types of simple movements: moving forward one step at a time; turning right 90 degrees; or turning left 90 degrees. You will also need to add a reporter that allows the turtle agent to sense whether there is food ahead.

Try to come up with a general solution, without resorting to the too specific will-only-work-on-the-Santa-Fe-Trail solution that the optimum solution represents. In other words, try to find a solution that has the best trade off between the number of steps taken by the program as it is executed times the size of the program code.

4.4 Drawing FSMs and Decision Trees using Link Agents in NetLogo

Exercise 4.4.1: Two States NetLogo Model

Try out the Two States Maze model in NetLogo:

Two States	http://files.bookboon.com/ai/Two-States.html

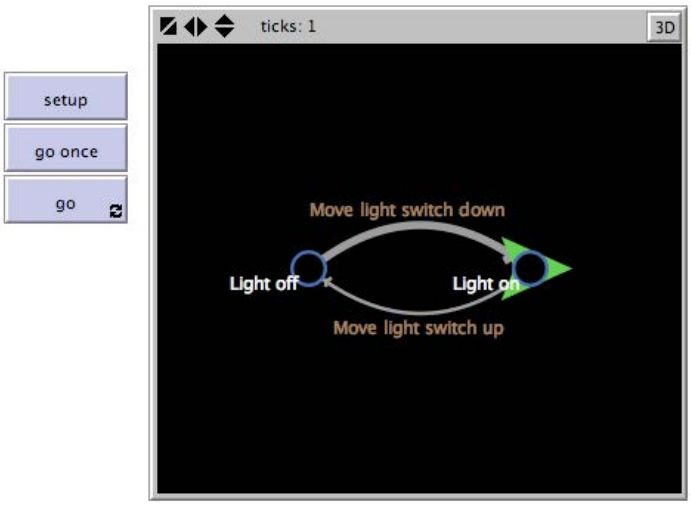


Figure 4.4.1. Screenshot of the Interface for the Two States model after the setup button has been pressed followed by the go once button.

WHAT IS IT?

This model shows how to draw a simple two-state Finite State Automata (FSA) that represents the process of turning a light switch off or on.

THE INTERFACE

The Interface buttons are defined as follows:

- `setup`: Sets up the FSA. (Note: sometimes this needs to be pressed twice in a row in order that the links between nodes become curved rather than straight).
- `go-once`: Moves the agent from one state to the other.
- `go`: The agent repeatedly moves back and forth between the states.

HOW IT WORKS

A single agent performs the animation. To simulate movement, the agent moves itself to the location of the adjacent state, and then increases the thickness of the link it is crossing over.

HOW TO USE IT

This model can be used as a starting point for drawing more complicated FSAs which have many more states and many agents moving around them.

WHAT IS ITS PURPOSE?

Its purpose is to show how you can animate FSAs in NetLogo.

EXTENDING THE MODEL

Try adding more states and more agents.

Try adding a state with a self-link back to itself. Hint: One way of doing this is to use a hidden state.

Exercise 4.4.2: Life Cycle Stages NetLogo Model

Try out the Life Cycle Stages model in NetLogo:

Life Cycle Stages	http://files.bookboon.com/ai/Life-Cycle-Stages.html

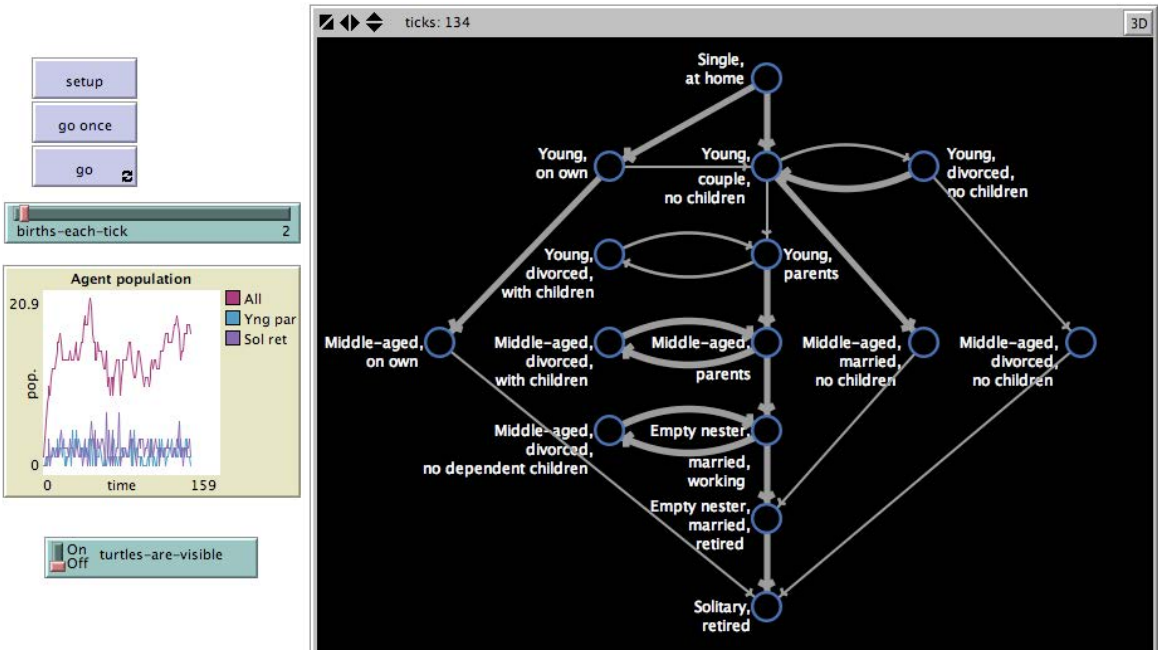


Figure 4.4.2. Screenshot of the Interface for the Life Cycle Stages model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model shows an example of finite state automata (FSA) that represents the life cycle stages of people throughout their lives. The model is based on a model proposed by D. Jobber (1998) in "*Principles and Practice of Marketing*", McGraw-Hill. The FSA is drawn rotated 90 degrees to the way Jobber draws it. This is simply because it is easier to fit the middle line of states vertically rather than horizontally within the NetLogo environment.

The model illustrates that life is not a linear progression – there are cycles that people go through.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This sets up the FSA.
- `go-once`: This executes one tick of the simulation.
- `go`: This makes the simulation runs continuously.

The `Interface` sliders and switches are defined as follows:

- `births-each-tick` slider: This is the number of turtle agents that are born each tick (that start at the "`Single, at home`" state at the top middle of the FSA.
- `turtles-are-visible` switch: If this is set to `Off`, the turtles will become invisible.

The `Interface` plot labelled "`Agent population`" plots the number of agents that currently exist versus time tick for three sets of states: for all states; for just the "`Young, parents`" state; and for just the "`Solitary, retired`" state.

HOW IT WORKS

The slider `birth-each-tick` controls the number of turtle agents that are born each tick. They will start out in the top middle state labelled "`Single, at home`". Each subsequent tick, these agents will randomly choose to head to one of the outgoing states. Eventually, each agent will end up in the final state labelled "`Solitary, retired`" at the middle bottom.

HOW TO USE IT

This model is another more complicated example of a FSA. It can be used as a starting point for drawing other FSAs, and then animating them. To run the simulation, press the `setup` button first, followed by either the `go once` button (to make the simulation proceed one tick) or the `go` button (to make the simulation proceed continuously).

WHAT IS ITS PURPOSE?

Its purpose is to show how you can animate FSAs in NetLogo.

RELATED MODELS

See the Two States models.

Exercise 4.4.3:

Use the sample code in the two models above (Two States and Life Cycle Stages) to create your own animated FSM in NetLogo (it must contain at least three states).

Exercise 4.4.4: NZ Birds Model

Try out the NZ Birds model in NetLogo:

NZ Birds	http://files.bookboon.com/ai/NZ-Birds.html

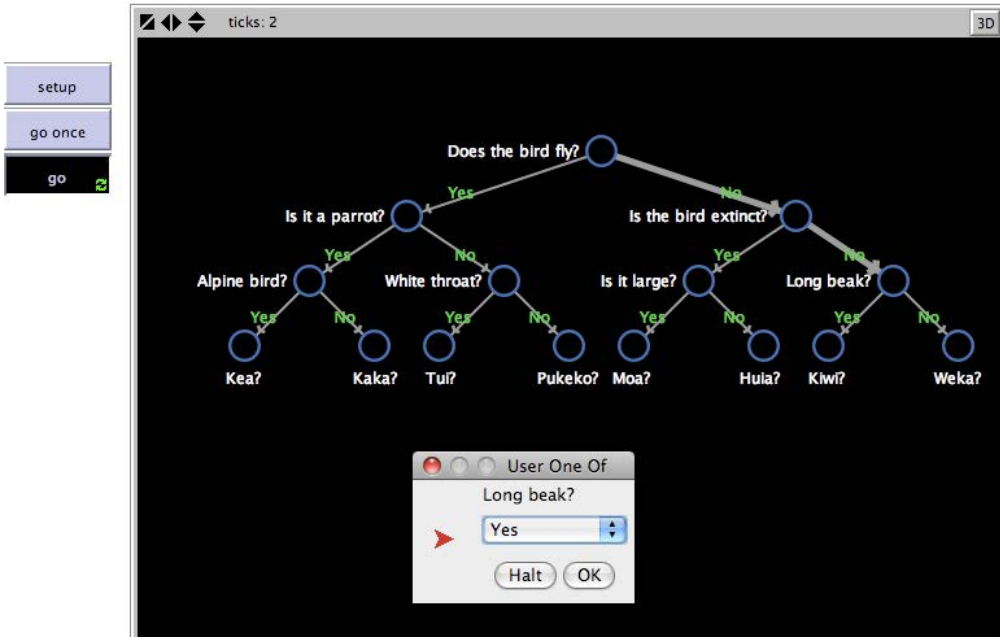


Figure 4.4.4. Screenshot of the Interface for the NZ Birds model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model shows how to create a decision tree in NetLogo and then animate it. The decision tree consists of a series of questions that can be used to help guess whether a bird a person is thinking of is a certain type of New Zealand bird or not (much in the same manner as the parlour game Twenty Questions).

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This sets up and displays the entire decision tree.
- `go once`: A single question is asked based on the current point in the decision tree.
- `go`: The questioning process continues until an answer is found.

HOW IT WORKS

The states of the decision tree are represented by turtle agents (called "`points`" in the model) and these contain a question variable that is the question to be asked of the walker agent at a decision point. The walker agents are represented by turtle agents (called "`agents`" in the model) and these store the location variable which is set to the point in the decision tree which the agent can currently be found. Answers are stored as variables associated with the links between the nodes in the decision tree (i.e. the `points`). The walker agents move to the child node depending on the answer that the user responds with to the question.

HOW TO USE IT

Ask someone to think of a New Zealand bird, then run the model by pressing the `setup` followed by the `go` buttons to see if the model can guess which bird they were thinking of.

WHAT IS ITS PURPOSE?

Its purpose is to illustrate how you can create a decision tree in NetLogo and animate it. A secondary purpose is to show that decision-making of an agent is analogous to movement in an environment.

THINGS TO TRY

See what happens when the person you are playing with chooses a non-New Zealand bird to play with.

See what happens when the person you are playing with gives an incorrect answer or does not know the answer to the question.

EXTENDING THE MODEL

Try creating a different decision tree.

Try adding a "don't know" option to the possible answers to the questions.

4.5 Computer Animation

Exercise 4.5.1:

Try out the Shape Animation model in NetLogo:

Shape Animation Example	NetLogo Models Library: Code Examples > Shape Animation Example http://ccl.northwestern.edu/netlogo/models/ShapeAnimationExample

Explain what the following procedures do in this model:

- `animate;`
- `move-person;`
- `age-flower;`
- `make-flower.`

Exercise 4.5.2: Stick Figure Walking Model

Try out the Stick Figure Walking model in NetLogo:

Stick Figure Walking	http://files.bookboon.com/ai/Stick-Figure-Walking.html

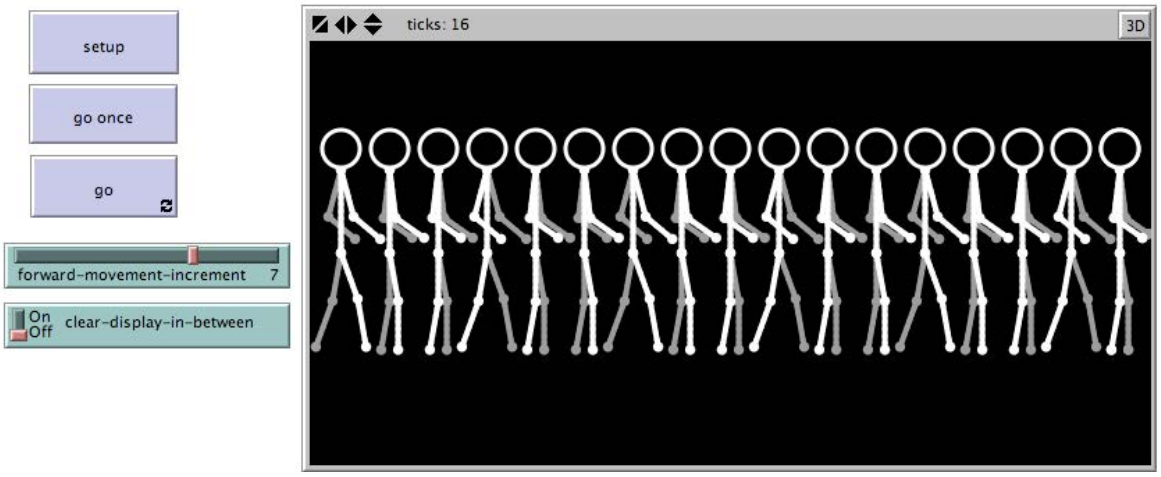


Figure 4.5.2. Screenshot of the Interface for the Stick Figure Walking model after the setup button has been pressed followed by the go button.

WHAT IS IT?

This model shows how to get a stick figure to walk across the screen.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This resets the animation by placing the stick figure at the left of the environment.
- `go-once`: The stick figures move across the screen once from left to right.
- `go`: The stick figure repeatedly walks across the screen from left to right.

The `Interface` sliders and switches are defined as follows:

- `forward-movement-increment`: This is how far the stick figure moves forward before the next frame is drawn.
- `clear-display-in-between`: If this is set to `Off`, all the frames will be drawn without the previous one being cleared first.

HOW IT WORKS

It uses six versions of the stick figure to perform basic key-frame animation. There are six configurations of the stick figure that correspond to the six frames used to define the walking motion. These six frames are displayed one after the other, and then the animation repeats. i.e. The frame sequence is 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, …

The stick figure is drawn using two turtle breeds – one for the head, and the other for the body and limbs. This is done using NetLogo turtle drawing commands rather than using link agents.

HOW TO USE IT

You can't really use it for anything, except for changing the values in the sliders to create different sized mazes. Press the `setup` button first, then if you want to run the animation sequence once, then press the `go once` button. If you want the animation sequence to run continuously, press the `go` button. Changing the `forward-movement-increment` slider changes the speed of the movement if the `clear-display-in-between` switch is set to `On`. If set to `Off`, it draws all the frames.

WHAT IS ITS PURPOSE?

Its purpose is to show how to do basic key-frame animation, and also show how changing a single variable that specifies the forward movement of the stick figure (e.g. the `forward-movement-increment` variable) can yield a surprising range of movements.

A further purpose is to illustrate how movement is fundamental to an agent in an environment and can be used to distinguish its type of behaviour. The movement of the stick figure's body & head (a manifestation of its embodiment) can also be decomposed down to movement of individual parts of its body & head.

THINGS TO TRY

Try changing the `forward-movement-increment` slider to see what happens to the animation. Try doing this dynamically, for example, by changing the slider during the middle of an animation. This will result in the stick figure seeming to slow down to a walk or speed up to a run. If you set the increment to negative, it will start moving backwards. If you set the increment to -1, you can also get the stick figure to moonwalk.

EXTENDING THE MODEL

Try using a different set of frames to get the stick figure to walk/run/jump/crawl in different ways.

RELATED MODELS

See the Stick Figure Animation model.

Exercise 4.5.3: Stick Figure Animation Model

Try out the Stick Figure Animation model in NetLogo:

Stick Figure Animation	http://files.bookboon.com/ai/Stick-Figure-Animation.html

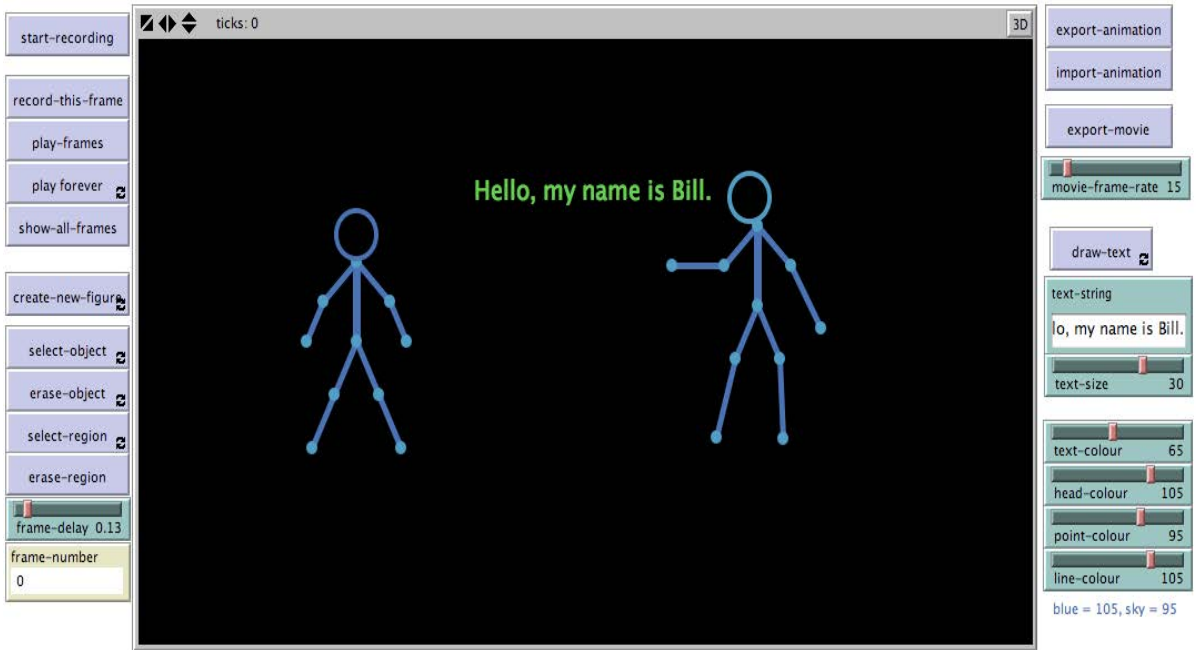


Figure 4.5.3. Screenshot of the Interface for the Stick Figure Animation model after: first, the start-recording button has been pressed; then second, two stick figures have been added to the environment using the create-new-figure button, with the right one altered using the select-object and select-region buttons; and then third, some text added using the draw-text button.

WHAT IS IT?

This model allows you to create your own stick figure animations and export them to files for importing latter on, or export them to QuickTime movie files. The model was inspired by the Pivot Stick Figure Animator application created by Peter Bone.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `start-recording`: This resets the animation sequence.
- `record-this-frame`: This takes a snapshot of whatever is currently on the screen, and adds it as a next frame to the animation sequence.
- `play-frames`: This will play (once) all the frames recorded in the animation sequence. The delay between playing each frame in the sequence is specified by the `frame-delay` slider. Which frame is being played is shown by the `frame-number` monitor.
- `play forever`: This will play the recorded frames repeatedly.

- `show-all-frames`: This shows all recorded frames superimposed on top of one another with their frame numbers shown.
- `create-new-figure`: This draws a new stick figure on the screen at the mouse's current position when the mouse is next clicked. The colours of the stick figure are specified by the `head-colour`, `point-colour` and `line-colour` sliders.
- `select-object`: This allows the user to select an object (the head of a stick figure or one of its pivot points) so that it can be moved.
- `erase-object`: This allows the user to erase an object.
- `select-region`: This allows the user to select multiple objects inside a rectangular region in order to move them.
- `erase-region`: This allows the user to erase objects inside a region.
- `export-animation`: This exports the animation to a file on disk so that it can be imported at a latter date. (Note: This will not work from an applet. Download the model file, and run it directly).
- `import-animation`: This imports the animation from a file already saved on disk. (Note: This will not work from an applet. Download the model file, and run it directly).
- `export-movie`: This exports the animation to a QuickTime movie file. The movie frame rate is specified by the `movie-frame-rate` slider. (Note: This will not work from an applet. Download the model file, and run it directly).
- `draw-text`: This allows the user to draw some text at the mouse's current position when the mouse is next clicked. The text that is drawn is specified by the `text-string` box and its font size and colour is specified by the `text-size` and `text-colour` sliders.

HOW IT WORKS

Each stick figure is drawn using three breeds of agents – a `head` turtle agent for representing the head; a `point` turtle agent for representing the stick figure's pivot points; and a `line` link agent for representing the body and limbs. In addition, a `text` turtle agent breed is used for drawing the text.

Recording is done by making use of NetLogo's `show-turtle` and `hide-turtle` commands. Only the current frame's agents are shown – the other recorded agents are hidden. Copies of the turtle agents and link agents from all the old recorded frames remain in the environment. These copies are stored in specific recording agent breeds – `recorded-heads`, `recorded-points`, `recorded-lines` and `recorded-texts` – so that the current frame's agents can easily be distinguished from the recorded frame's agents. These recorded agents also have an additional variable – `frame`, `hframe`, `lframe` or `tframe` – that is used to store the frame number when they were recorded.

When the animation is played back, the procedure executes a while loop for each frame number. This procedure repeatedly hides all agents first, then only the recorded agents with the current frame number are shown.

HOW TO USE IT

Use the `start-recording` button to start recording the animation sequence. Add as many new stick figures as you want using the `create-new-figure` button. Move these figures around or delete them using the editing buttons – `select-object`, `erase-object`, `select-region` and `erase-region`.

Each time you want a particular screen to be recorded for the animation, press the `record-this-frame` button. Nothing is recorded until this button is pressed – the movement of the objects around while they are being edited will not end up being recorded in the animation sequence. You can play the animation sequence once by pressing the `play-frames` button, or have it loop continuously by pressing the `play forever` button.

When you are satisfied with the animation, either save it to disk using the `export-animation` button so that it can loaded latter using the `import-animation` button, or export it to a QuickTime movie using the `export-movie` button.

WHAT IS ITS PURPOSE?

This model has been inspired by the endless variety and sheer creativity of the countless stick figure animations that can be found in YouTube. The style of the model is reminiscent of the public domain Pivot Stick Figure Animator program created by Peter Bone.

Its purpose is to show how to do basic key-frame animation. A further purpose is to illustrate how movement is fundamental to an agent in an environment and can be used to distinguish its type of behaviour. The movement of the stick figure's body & head (a manifestation of its embodiment) can also be decomposed down to movement of individual parts of its body & head.

THINGS TO TRY

Try creating elaborate stick figure animations. You are only limited by your imagination. If you get stuck, look at some stick figure animations on YouTube.

EXTENDING THE MODEL

Try allowing the user to add other objects such as trees, cars and buildings.

A quick look in YouTube will show that one particular variety of animation occurs frequently –stop-frame animation using Lego characters, especially Lego Star Wars. Alter this model to show Lego figures as well as stick figures (such as Lego Clone Troopers and a Lego Yoda).

Once this is done, then you can try creating an alternate Star Wars universe where the familiar Star Wars characters fight against the invading Stick Figures.

RELATED MODELS

See the Stick Figure Walking model.

Exercise 4.5.4:

Find out what the Stick Figure Animation model does when the following buttons are pressed:

- `record-this-frame;`
- `play-frames;`
- `show-all-frames;`
- `export-animation;`
- `import-animation;`
- `export-movie.`

Exercise 4.5.5:

Write a NetLogo model that will animate the simple movement of an agent across the environment from left to right like the Stick Figure Walking model. Add a chooser into the `Interface` so that the user can select the shape of the agent that is animated – for example, cars, trucks, trains and buses.

4.6 Animated Mapping and Simulation

Exercise 4.6.1: Continental Divide NetLogo Model

Try out the Continental Divide model in NetLogo:

Continental Divide	From NetLogo's Model Library: Earth Science > Continental Divide http://ccl.northwestern.edu/netlogo/models/ContinentalDivide

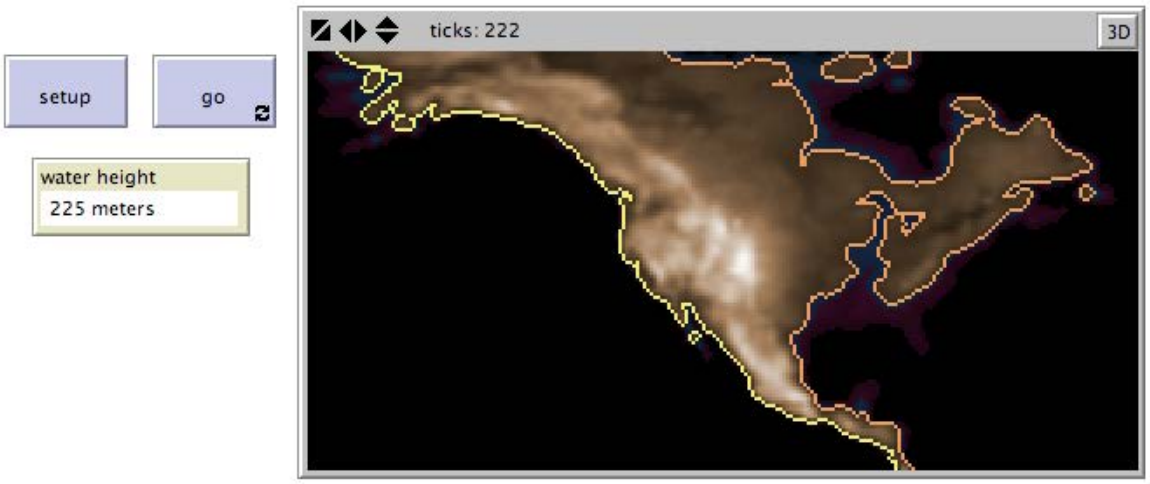


Figure 4.6.1. Screenshot of the Interface for the Continental Divide model after the setup button has been pressed followed by the go button.

Find out how the model sets up the environment from the elevation data (note the use of the `foreach` command on multiple lists).

Also find out how the flood is performed.

Exercise 4.6.2: Grand Canyon NetLogo Model

Try out the Grand Canyon model in NetLogo:

Grand Canyon	From NetLogo's Model Library: Earth Science > Grand Canyon http://ccl.northwestern.edu/netlogo/models/GrandCanyon

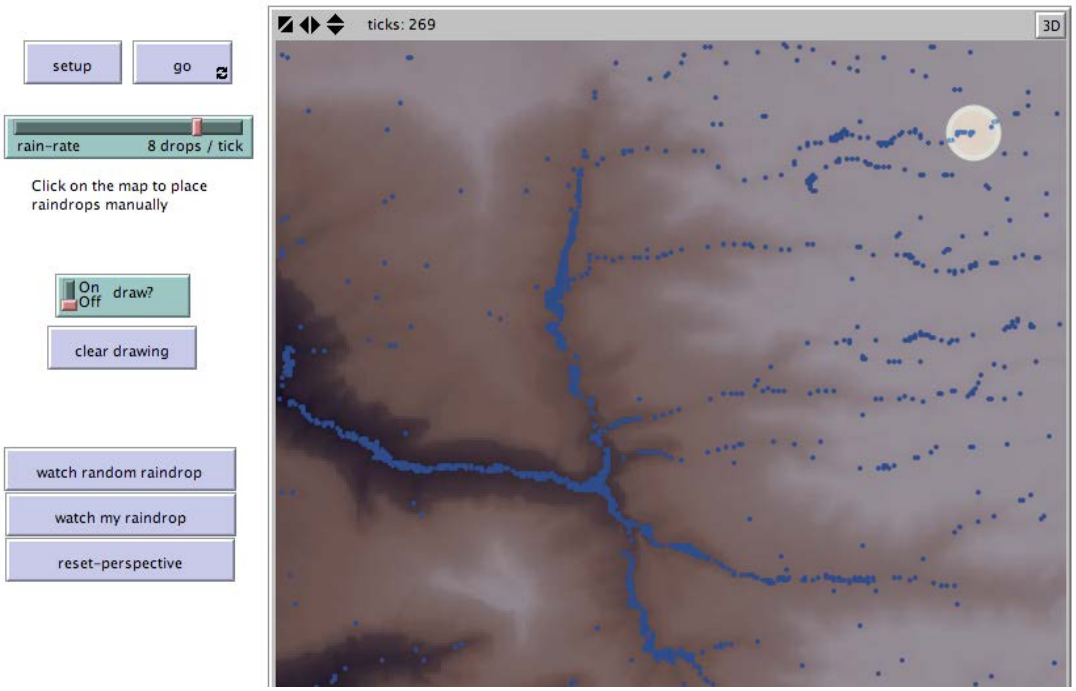


Figure 4.6.2. Screenshot of the Interface for the Grand Canyon model after the setup button has been pressed followed by the go button. A halo is shown round one of the raindrops when the watch random raindrop button is pressed, as in the image.

How is the elevation data imported into this model?

How does the `flow` procedure work? In other words, how do the `raindrops` agent breed behave in order to simulate the flow of water? Note at one point in the procedure that the breed of the `raindrops` agents is switched to the `waters` breed. Why is this done?

Note the use of the global `border` agentset of patches to simplify the recognition of when the edge of the environment has been reached.

5 Embodiment

5.1 Our body and our senses

Exercise 5.1.1:

Try adding your own examples of Body or Mind, Sight, Hearing, Smell, Taste and eating, or Touch and feeling based metaphors to Table 5.1 from Chapter 4 of the companion book "*Artificial Intelligence – Agents and Environments*".

5.2 Several Features of Autonomous Agents

Exercise 5.2.1:

Fill in the following table using several examples of the different types of agents from real and virtual worlds listed in the first column:

Type of agent	Embodiment (sensing capabilities)	Environment	Behaviour (reactive or cognitive)
Real-life agent			
Real autonomous robot			
Real semi-autonomous robot			
Virtual agent			
Virtual semi-autonomous agent			
Simulated agent			

For the second column, describe the embodiment and sensing capabilities that the agent uses to interact with and sense the world. For the third column, describe the environment that the agent is situated within, whether real, virtual or abstract. Fill in the fourth column with details of its behaviour, and whether the behaviour is mostly reactive or more cognitive.

5.3 Adding Sensing Capabilities to Turtle Agents in NetLogo

Exercise 5.3.1:

Try out the following models in NetLogo using the links below. These models implement basic sensing capabilities in turtle agents. These models (apart from the Wall Following Example model) have been updated from the ones provided in the Code Examples of NetLogo's Models Library by adding further sliders in the `Interface` to extend the simulation options as detailed below.

The Look Ahead Example 2 model simulates a basic sense of vision for each turtle agent by using the `patch-ahead` command. The model has been modified to add the `number-of-turtles` slider in the `Interface` to allow the user to control how many turtles are created at the start of the simulation.

Look Ahead Example 2	In NetLogo Model's Library: Code Examples > Look Ahead Example. See modified code at: http://files.bookboon.com/ai/Look-Ahead-Example-2.html

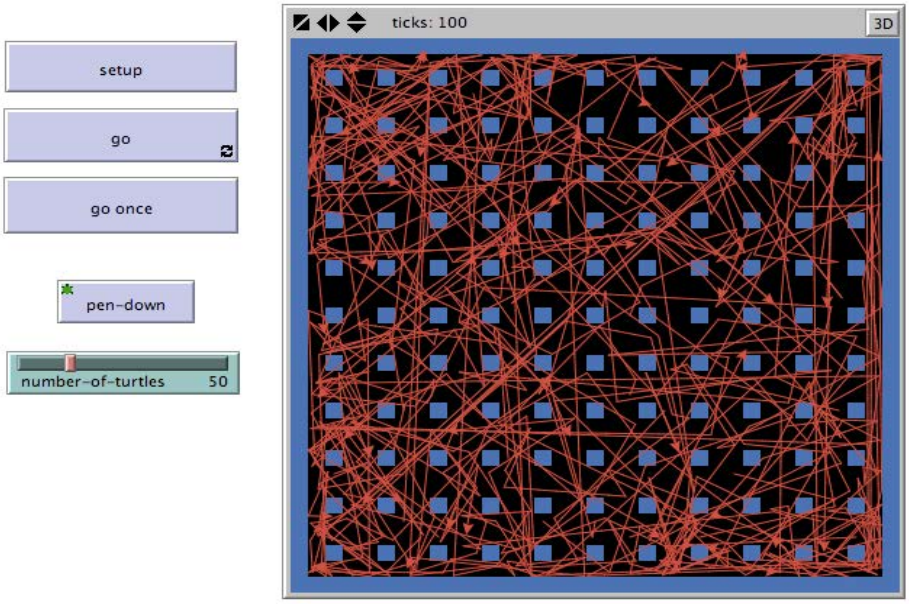


Figure 5.3.1.1 Screenshot of the Interface for the Look Ahead Example 2 model after the setup **button has been pressed followed by the** pen-down **and** go **buttons.**

The Wall Following Example model is the same as the one in NetLogo's Models Library. It demonstrates wall following behaviour by using the `patch-right-and-ahead` command to simulate the sense of touch via proximity detection.

Wall Following Example	In NetLogo Model's Library: Code Examples > Wall Following Example http://ccl.northwestern.edu/netlogo/models/WallFollowingExample

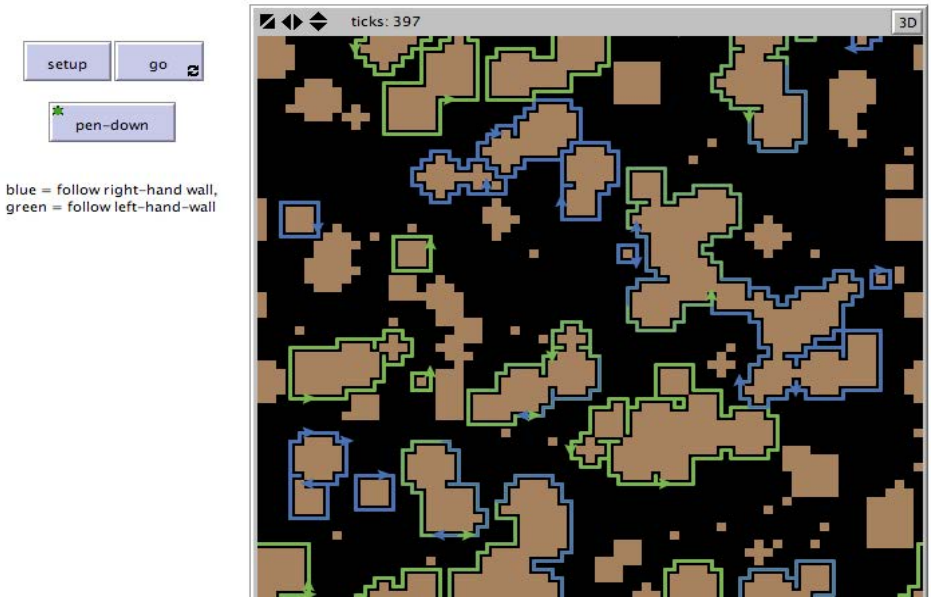


Figure 5.3.1.2 Screenshot of the Interface for the Wall Following model after the setup button has been pressed followed by the pen-down and go buttons.

The Line of Sight Example 2 model is a slight modification of the model provided in NetLogo's Models Library – instead of 6 `walker` turtle agents being created, there are 20. The model implements a sense of vision by working out what each `walker` agent would "see" in its line of sight. Sometimes due to the immediate terrain, this line of sight may be restricted because of nearby hill obscuring its vision. Other times, the line of sight will be longer.

Line of Sight Example 2	In NetLogo Model's Library: Code Examples > Line of Sight Example. See modified code at: http://files.bookboon.com/ai/Line-of-Sight-Example-2.html

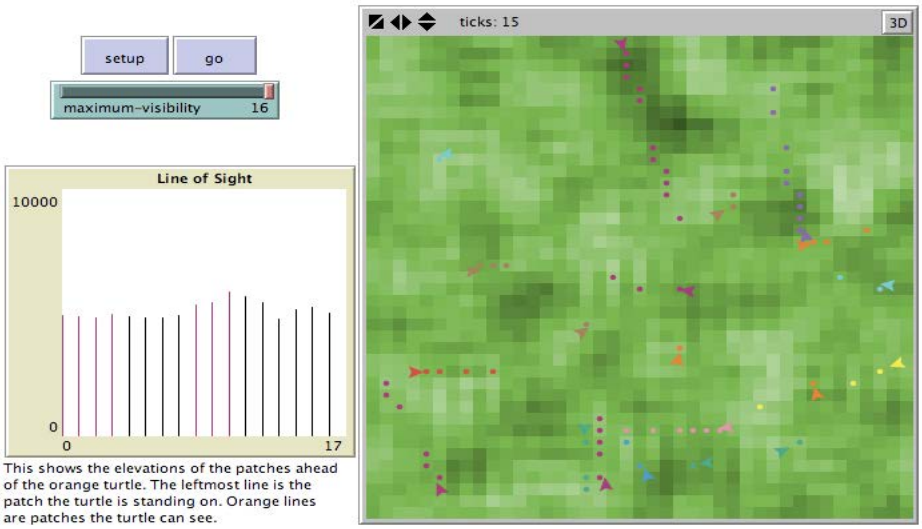


Figure 5.3.1.3 Screenshot of the Interface for the Line of Sight Example 2 model after the setup button has been pressed followed by the go button.

The Vision Cone Example 2 model is a slight modification of the model provided in NetLogo's Models Library. Two sliders have been added to the `Interface` – `number-of-standers` and `number-of-wanderers` – to control the number of grey `stander` agents that don't move, and the number of red `wanderer` agents that move around sensing what is in the environment. The model implements the `wanderer`'s sense of vision using NetLogo's `in-cone` command.

Vision Cone Example 2	In NetLogo Model's Library: Code Examples > Vision Cone Example. See modified code at: http://files.bookboon.com/ai/Vision-Cone-Example-2.html

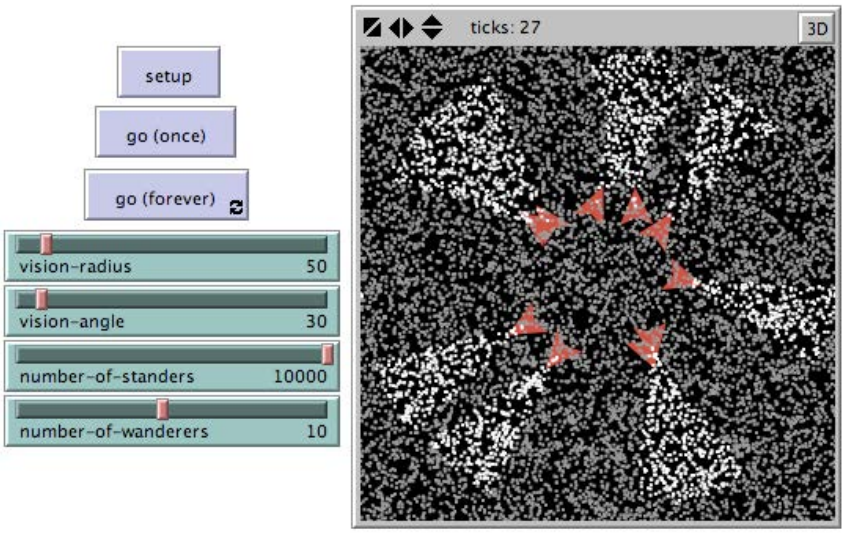


Figure 5.3.1.4 Screenshot of the Interface for the Vision Cone Example 2 model after the setup button has been pressed followed by the go(forever) button.

The Hill Climbing Example 2 model simulates turtle agents with an ability to sense what is uphill in the terrain using NetLogo `uphill` command. The model is the same as provided in NetLogo's Models Library, except for the following added to the `Interface`: a `go once` button so that the user can control the movement of the turtle agents one step at a time per tick; and the `number-of-turtles` slider that controls the number of turtles created in the environment. Since turtles are created at random patches using the `n-of` and `sprout` commands, increasing the value of this slider to greater than or equal to the number of patches, followed by pressing the `go` button will result in all the possible paths the turtle agents can take (as dictated by its hill-climbing behaviour), being drawn in the environment, as shown in the image below. This results in an almost 3D effect where the paths along the highest points of the ridge-lines are clearly shown.

Hill Climbing Example 2	In NetLogo Model's Library: Code Examples > Hill Climbing Example. See modified code at: http://files.bookboon.com/ai/Hill-Climbing-Example-2.html

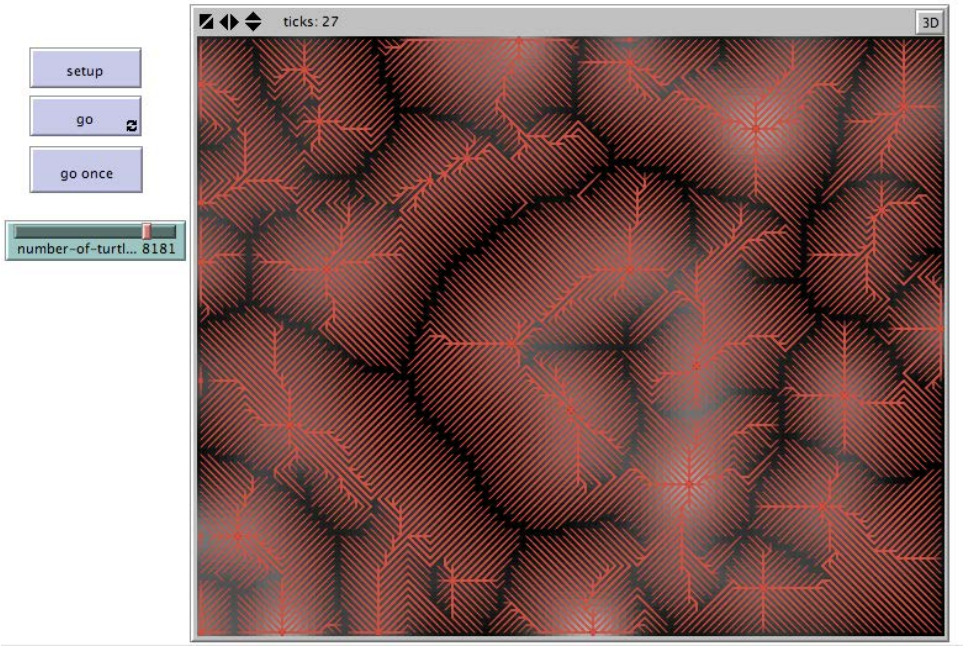


Figure 5.3.1.5 Screenshot of the Interface for the Hill Climbing Example 2 model after the setup button has been pressed followed by the go button. The case shown occurs when the value in the number-of-turtles slider exceeds the maximum number of patches in the environment.

Analyse each of the models to see how effective they are for the turtle agents at providing a means for sensing what is happening in the environment. Have a go at further extending the models by adding further capabilities to enhance the turtle agent's senses. How difficult would it be to have a turtle agent have more than once sense? That is, what are the problems at coordinating the actions of the agents in response to sensory events happening on multiple input streams?

Exercise 5.3.2:

When the `pen-down` button is pressed in the Look Ahead Example 2 model, the paths quickly fill up the rest of the environment apart from the blue patches. Make use of this behaviour to design a model that simulates an autonomous vacuum cleaner robot that cleans the floor of a room (like the one depicted in Section 2.2 of the *Artificial Intelligence – Agents and Environments* book). Try adding one or more further behaviours so that the simulated robot covers the environment in a different manner to the method used for the Look Ahead Example 2 model. Extend the model so that the user can add further obstacles into the environment.

5.4 Performing tasks reactively without cognition

Exercise 5.4.1: Ants NetLogo Model

Try out the Ants model in NetLogo:

Ants	In NetLogo's Models Library: Biology > Ants http://ccl.northwestern.edu/netlogo/models/Ants

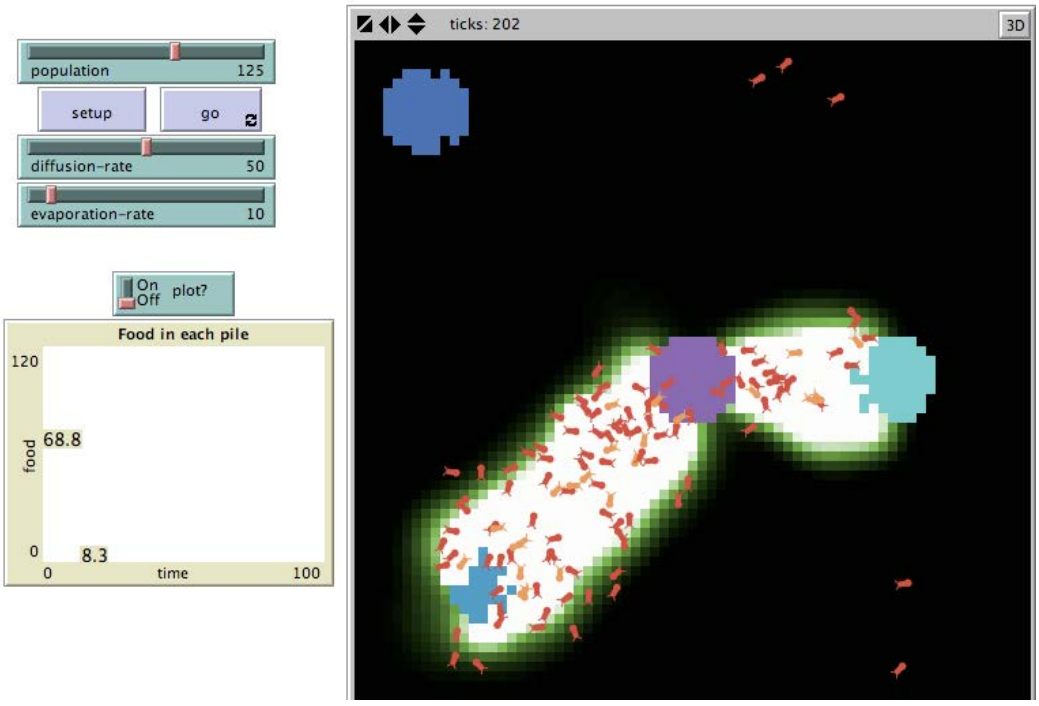


Figure 5.4.1. Screenshot of the Interface for the Ants model after the setup button has been pressed followed by the go button.

Identify how the following patch variables are used in this model:

- `chemical;`
- `food;`
- `nest?;`
- `nest-scent;`
- `food-source-number.`

Also find out how the following procedures work:

- `look-for-food;`
- `return-to-nest;`
- `uphill-chemical;`
- `uphill-nest-scent.`

Exercise 5.4.2: Mazes NetLogo Model

Try out the Mazes model in NetLogo:

Mazes	http://files.bookboon.com/ai/Mazes.html

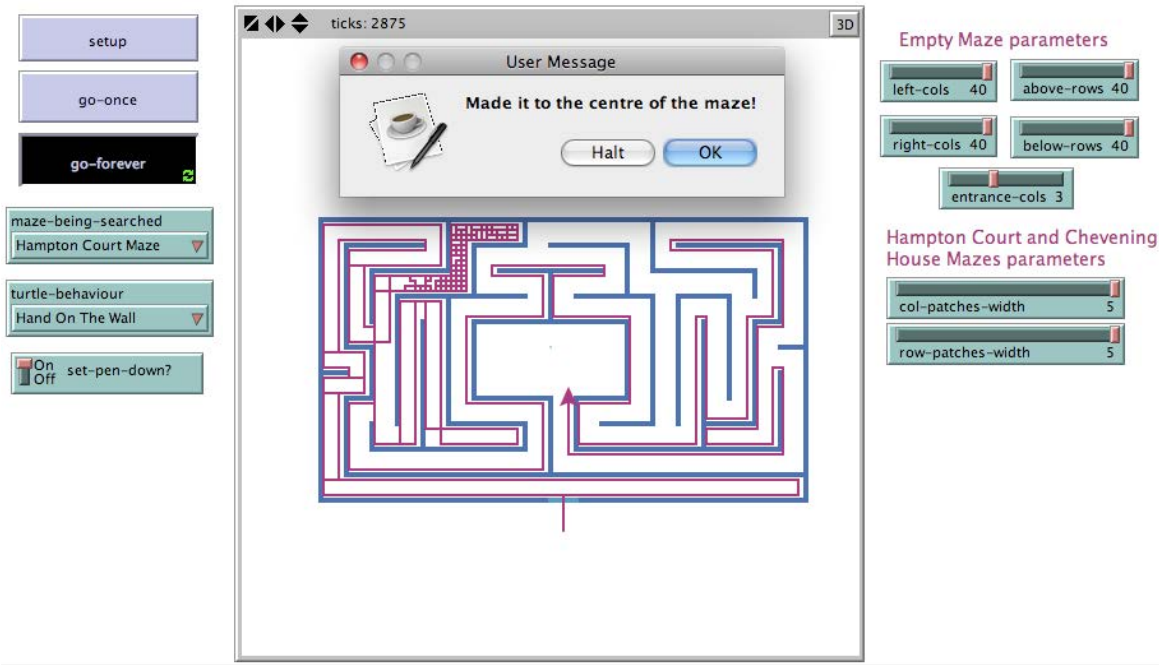


Figure 5.4.2. Screenshot of the Interface for the Mazes model after the setup button has been pressed followed by the go-forever button. The chosen maze is the Hampton Court Maze, and the order chosen for the turtle behaviour as the turtle was moving around the maze was as follows: "Random Forward 1", then "Random Forward 2", then "Hand On The Wall".

WHAT IS IT?

This model shows how to get a simple reactive turtle agent to move around a maze. The model comes with three mazes – the empty maze (just outside walls), and two mazes that are schematic representations of real life mazes in the United Kingdom – the Hampton Court Palace maze, and the Chevening House Maze.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `setup`: This will clear the environment and redraw the maze selected by the `maze-being-searched` chooser.
- `go-once`: This will get a single turtle agent to walk around the maze according to the behaviour selected in the `turtle-behaviour` chooser. This button will execute the `walk` turtle procedure once, not continuously.
- `go-forever`: This will get a single turtle agent to continuously walk around the maze according to the behaviour selected in the `turtle-behaviour` chooser.

The choosers, sliders and switch are defined as follows:

- `maze-being-searched`: This specifies the maze that is being searched. This is either the empty maze, the Hampton Court Palace maze, or the Chevening House maze.
- `turtle-behaviour`: This specifies the type of reactive behaviour the walker turtle agent exhibits. The types of behaviour is as follows:

  `"hand-on-the-wall"`: This is the classic maze wall following behaviour of keeping a specific hand, either left or right, on a wall at all times. The model randomly decides itself whether the left hand or right hand is chosen.

  `"Random Forward 0"`: In this behaviour, the walker turtle agent moves forward unless there is a wall ahead, then tries to turn left unless there is a wall to the left, then tries to turn right unless there is a wall to the right, then randomly turns left or right as a last resort.

  `"Random Forward 1"`: In this behaviour, the walker turtle agent moves around mostly in straight lines in random directions for a random distance, but if it encounters a wall, it will bang against it for a short while (much like a woodpecker banging against a tree or a fly against a window) before backing off a random distance and continuing on with its random wanderings.

  `"Random Forward 2"`: In this behaviour, the walker turtle agent wanders around in random directions using small steps.

- `set-pen-down?`: If set to `On`, this will draw the path of the walker turtle agent.
- `left-cols`, `right-cols`, `above-rows`, `below-rows`, `entrance-cols`: This sets the length and widths of the empty maze's side walls and entrance.
- `col-patches-width`, `row-patches-width`: This sets the width in patches for the columns and rows of the Hampton Court maze and Chevening House maze (i.e. the width between horizontal and vertical walls in these mazes).

HOW IT WORKS

It uses one `ask patches` command to set the patches blue that define the walls, and to set the remaining patches white. The code uses two procedures that are similar, `setup-row` and `setup-col`, to draw the walls in a horizontal row or vertical column respectively. These procedures take the row or column number as the first parameter, the colour that the walls are to be drawn with, and a list containing two-numbered range lists that define the segments where the walls are to be drawn in the row or column.

One turtle agent is created as the walker agent. It uses a simple proximity detector method (see the `wall?` reporter) to sense if there is a wall ahead at a specific angle and distance.

HOW TO USE IT

First select the maze using the `maze-being-searched` chooser, then select the behaviour of the turtle using the `turtle-behaviour` chooser. Then press the `setup` button, followed by either the `go-once` or `go-forever` buttons.

To get the turtle to draw its path, set the `set-pen-down?` switch to `On`.

WHAT IS ITS PURPOSE?

Its purpose is to show how a simple reactive agent can be effective (or not) at exploring a maze. The turtle has no cognitive abilities to recognize the situation it is in; e.g. that it is in a maze, that there is a goal it should try to reach, that it gets stuck at times, that there are alternative paths it can take at various points, or that it has travelled over the same point multiple times. The turtle agent simply reacts to the immediate situation it finds itself in using a simple proximity detector to sense a wall nearby.

The model also uses two virtual mazes that have a corresponding maze in real-life to demonstrate how virtual environments can mirror real-life environments but where the reflection can often be distorted in the process.

THINGS TO TRY

Try out the different behaviours on the different mazes by changing the values of the `maze-being-searched` and `turtle-behaviour` choosers. Try doing this as the turtle is moving around.

See what happens when you change the value of the sliders.

Try changing the `Settings` of the environment such as the `Patch Size` and the maximum and minimum x and y co-ordinates.

EXTENDING THE MODEL

Try adding your own behaviours to get the turtle agent to move around the maze.

NETLOGO FEATURES

The model uses the `path-right-and-ahead` reporter to allow the turtle agent to sense if there is a wall nearby.

RELATED MODELS

See the Empty Maze, Hampton Court Maze and Chevening House Maze models.

Exercise 5.4.3:

Change the shape of the turtle that searches the maze to a `mouse` shape. Do this by selecting the `Import from Library` option in the `Turtle Shapes Editor`.

Also try changing the size of the turtle to see what happens within the different environments when the turtle is larger or smaller.

Exercise 5.4.4:

Add the maze your created yourself in Exercise 3.7.3 to the Mazes model. How do the turtles with the different turtle behaviours cope with the new environment? Is there something different about your maze compared to the other three mazes (the empty maze, the Hampton Court Palace maze, and the Chevening House maze) that results in novel behaviour?

Also try adding your own turtle behaviours.

Exercise 5.4.5:

How do the following procedures work that define the different behaviours of the turtle for the Mazes model?:

- `behaviour-wall-following;`
- `behaviour-random-forward-0;`
- `behaviour-random-forward-1;`
- `behaviour-random-forward-2.`

Exercise 5.4.6:

Try to reproduce what is seen in the screenshots for the Mazes model that are shown in Figures 5.8, 5.9 and 5.10 of the book '*Artificial Intelligence – Agents and Environments*'. For most of these, it is virtually impossible to reproduce exactly what is seen in the screenshots. Why? Which ones can be reproduced exactly?

Exercise 5.4.7:

Try to get the turtle to reach the centre of the Chevening House Maze. You will need to do this by using the wall following behaviour most of the time, but temporarily switching to another behaviour to jump from one island to another. Which combination of behaviours seem to be the most effective and at which points in the maze?

5.5 Embodied, Situated Cognition

Exercise 5.5.1:

Modify the behaviour of the turtle agent you devised for Exercise 4.3.1 for following the Santa Fe Ant Trail so that it uses a more sophisticated method to sense the presence of food. In Exercise 4.3.1, the turtle agent could only sense the presence of food directly ahead. Try different variations to the turtle agent's behaviour that takes into account its embodiment and situatedness. The idea here is to combine both sensing and motor operations together to perform a basic form of sensory-motor co-ordination. Compare the performance of the different ant behaviours you devised at following the ant trail.

(Hint: Consider the role that peripheral vision might have in improving the ant's abilities at following the trail).

Exercise 5.5.2:

Modify the model you developed for Exercise 5.5.1 so that the ant is able to follow any trail, not just the Santa Fe Ant Trail. Rather than have food deposited on the trail, assume the trail has been laid down by scent and the goal of the ant is to follow the chemical scent trail rather than find the food. Set up the initial conditions so that the ant starts at a random location, probably off the trail, rather than at the start of the trail as for Santa Fe Ant Trail problem. Hence the ant will have to add some mechanism for finding the trail to its behaviour. Then add multiple ants into your model to see what happens.

Exercise 5.5.3:

Devise a method for measuring which of the behaviours you developed for Exercises 5.5.1 and 5.5.2 are more effective for the ants in following the trails.

Exercise 5.5.4:

Devise a behaviour that will follow the Santa Fe Trail exactly without any further steps (i.e. devise an optimum solution). Record the number of steps the optimum solution takes compared to the other solutions you devised in Exercises 4.3.1 and 5.5.1. Also record the number of steps in the programs themselves (i.e. the size of the programs). Which behaviour was best?

Exercise 5.5.5:

Describe the advantages and disadvantages of the embodied approach. Your answer should include a discussion of the following claim:

"*The embodied approach is an interesting diversion from mainstream A.I. However to date, only basic systems have been built, and much more sophisticated algorithms are needed to develop it further so that it can tackle the deep problems of A.I. such as common sense reasoning, concept formation and language acquisition.*"

6 Solutions to Selected Exercises

Solution to Exercise 2.2.1:

There is no completely object-oriented solution to this problem as some agents (whether human or machine) have to be involved in performing this task in some manner.

One possible object-oriented solution is to use a standard car wash you see at a garage. However, this must be operated by a human agent and is therefore non-autonomous. The objects in this case are: the hoses and cleaning equipment; and the parameters are defined by the keypad used to operate the car wash.

One possible agent-oriented solution is to use an autonomous robotic car cleaner. This cleans the car like a human would. It makes its own decisions about how this should be done, and the manner and order the car should be washed. It also re-charges itself by finding a power outlet when it is running low on power. The objects are the cleaning equipment it uses. The agent is itself.

The car-wash solution above is an object-oriented solution because it has no control over when it is used and does not make decisions for itself. The autonomous robotic car cleaner solution is agent-oriented as it is deciding for itself how to perform the task.

Solution to Exercise 2.4.1:

This exercise highlights the problems with trying to categorize agent-oriented systems using labels such as "weak", "strong" and "strongest". As with all categorical definitions, real examples often refuse to be labelled as being in a single category (see Chapter 9 of the companion book *Artificial Intelligence – Agent Behaviour I* for further explanation). Many of the answers below indicate problems with this type of classification as most have both weak and strong properties.

Program	**Weak or strong agent?**
Google's Web crawler Googlebot.	This has weak agent properties plus aspects of a strong agent.
A program set up on a website to collect answers for a questionnaire.	This is not an agent (no autonomy for example).
A program for a supermarket to automatically locate and bid for the lowest food prices on its Extranet.	This is a strong agent (has weak plus strong agent properties above).
A distributed information retrieval program that helps you locate Web documents that you are interested in	This is a strong agent (has weak plus strong agent properties above).
A mail-filtering program that among other things remove SPAM messages from your email.	This is a weak agent (has weak agent properties above, but could also have some strong agent properties).
A multi-user Internet-wide game playing program.	This may be all three – not an agent if using client/server technology; or it might have both weak and/or strong agent properties).
A "chatterbot" program whose task is to send messages to chat-rooms and fool the humans into believing it is a real human and not a program.	This is a strong agent (has weak plus strong agent properties above).

Solution to Exercise 2.7.1

Descriptions of each of the images are as follows:

Top-left: This is a photorealistic computer-generated image using a persistence of vision ray tracer.

Top-right: This Nordic Bay lake sketch is an example of non-photorealistic architectural rendering.

Upper-middle-left: This is a pseudo-realistic example of fractal trees generated using L-systems.

Upper-middle-right: This is an image of the Opportunity Mars Exploration Rover created by NASA's Jet Propulsion Laboratory using a photo-realistic model of the rover and a false colour mosaic taken on June 9, 2004 by Opportunity's panoramic camera.

Lower-middle-left: This is a pseudo-realistic environment generated for Second Life.

Lower-middle-right: This is a pseudo-realistic environment generated using Ogre, the open source 3D graphics engine.

Bottom-left: This is an example of a pseudo-realistic scene from the orienteering sport simulation game Catching Features.

Bottom-right: This is another pseudo-realistic environment generated for Second Life.

Solution to Exercise 3.4.3:

NetLogo uses agent-directed simulation. `ask` commands are used to specify what each type of agent does (whether turtle, patch, link or observer). Arguably, this isn't really any different to a standard procedural programming language. At best, the agents are more 'proto-agents' with weak properties such as reactivity and temporal continuity with some sociability and partial autonomy but less of proactivity, rather than strong or strongest properties such as rationality, consciousness and self-awareness. i.e. they have reactive abilities rather than cognitive abilities.

Solution to Exercises 3.5.5 and 3.5.6:

The following NetLogo model provides a solution to these exercises:

Load File	http://files.bookboon.com/ai/Load-Files.html

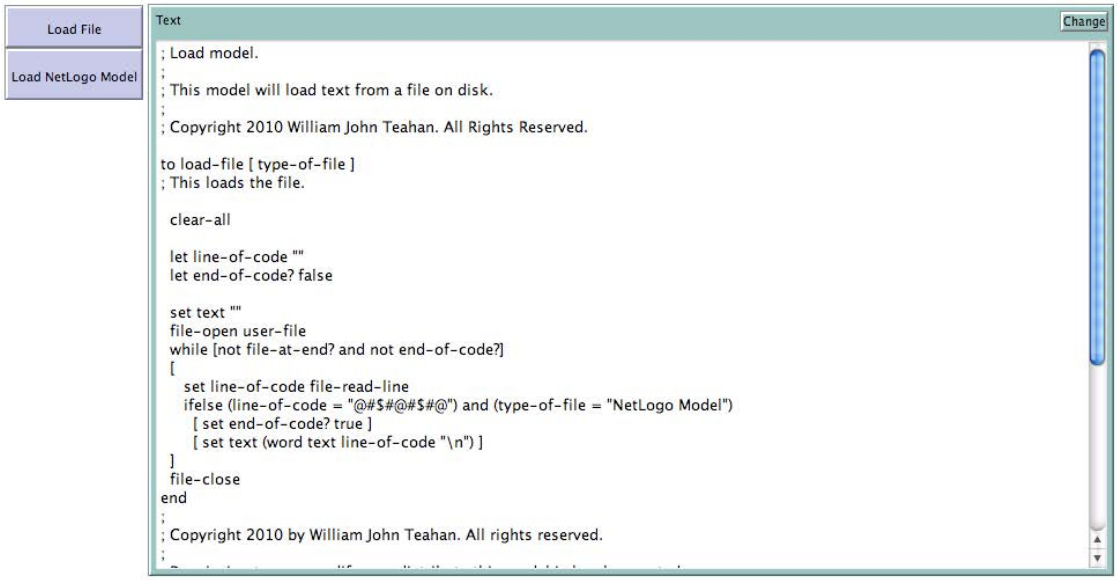


Figure S-3.5.3. Screenshot of the Interface for the Load File model after the Load NetLogo Model button has been pressed, and the Load-File.nlogo file has been loaded.

WHAT IS IT?

This model shows how to load text from a file.

WHAT IS ITS PURPOSE?

The model is a solution to Exercises 3.5.3 and 3.5.3 in the book "*Exercises for Artificial Intelligence – Agents and Environments*."

HOW IT WORKS

The procedure `load-file` reads each line of the input file one at a time, and appends it to a string variable called `text` which is displayed as an input box in the `Interface`. If the `type-of-file` parameter is set to `"NetLogo Model"`, it will search for the sequence of characters that delimits the end of the source code in NetLogo model files that have been saved to disk (stored in files with the `".nlogo"` extension).

HOW TO USE IT

Press the `Load File` to load a general text file, or `Load NetLogo Model` button to load a NetLogo model file. This will then open a dialog that allows the user to choose an existing disk file on the system. The text from the file will then be loaded directly into the `Text` input box.

Note that this model will not work from a Web browser when it has been saved as an applet as it needs to open a file.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `Load File`: This will load a text file from disk. This file can be any file on disk. If it is a NetLogo model file with the `".nlogo"` extension, it will load the full file rather than just the source code when the `Load NetLogo Model` button is pressed.
- `Load NetLogo Model`: This will load the source code for a NetLogo model text file from disk. This does not check whether the file really is a NetLogo model, however, so if it isn't, pressing this button will simply have the same effect as the `Load File` button.

THINGS TO NOTICE

Notice what happens when binary files, such as JPEG and GIF files, are loaded. Also notice what happens when you load a Word document.

THINGS TO TRY

Try loading the same NetLogo model using both the `Load File` and `Load NetLogo Model` buttons to see what happens.

Try loading other text files, such as a Word document or a HTML document to see what happens.

If a NetLogo model file has been loaded, see what happens if you add a button to run the code (using the `run` command).

EXTENDING THE MODEL

Try modifying the code so that it checks the type of file that is being loaded and rejects it if it isn't a NetLogo model or text.

NETLOGO FEATURES

Note the use of the `word` command to concatenate each input line onto the end of the text string.

Solution to Exercise 3.6.1:

This code creates 50 brown wolves and 500 white sheep at random locations in the environment. They would appear as arrowheads as no shape has been specified. Approximately 50% would be designated with gender = `"Male"` and the rest `"Female"`.

The turtles are the wolves and sheep breeds. The variables are `age`, `size`, `color`, `gender`.

Solutions to Exercises 3.6.7, 3.7.1, 3.7.2

For all of these exercises related to extending the models, look at the Mazes and Searching Mazes NetLogo models for solutions on how to design turtle agents to move around and search the mazes.

The three maze models have also been extended by adding a simple turtle agent with wall following behaviour. To try out these models, use the following links:

Empty Maze with Wall Following	http://files.bookboon.com/ai/Empty-Maze-with-Wall-Following.html
Hampton Court Maze with Wall Following	http://files.bookboon.com/ai/Hampton-Court-Maze-with-Wall-Following.html
Chevening House Maze with Wall Following	http://files.bookboon.com/ai/Chevening-House-Maze-with-Wall-Following.html

Solution to Exercise 3.6.8: Nested Squares NetLogo Model

The following NetLogo model provides a solution to this exercise:

Nested Squares	http://files.bookboon.com/ai/Nested-Squares.html

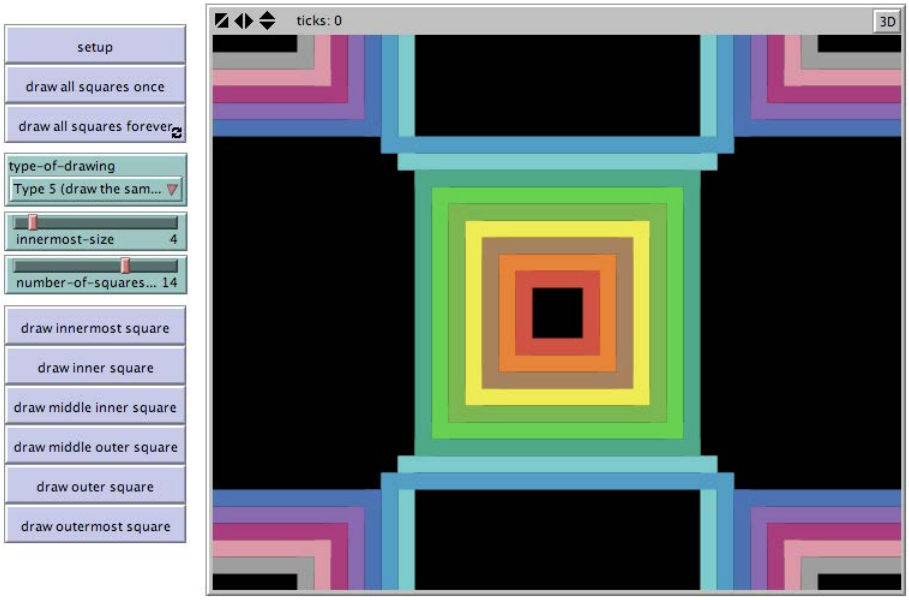


Figure S-3.6.8. Screenshot of the Interface for the Nested Squares model after the setup button has been pressed followed by the draw all squares once button, with the slider values as shown.

WHAT IS IT?

This model draws nested squares. In one configuration, it will draw 6 nested squares using 6 different methods illustrating the versatility of NetLogo for drawing. In another configuration, it will draw the nested squares using the same method.

The six different methods for drawing the squares are as follows:

Type 1. This draws the square using a single turtle agent with the shape `"square 2"`.
Type 2. This draws the square using a single turtle agent with the `setxy` command.
Type 3. This draws the square using a single turtle agent and the `stamp` command.
Type 4. This draws the square using multiple turtle agents.
Type 5. This draws the square using four turtle agents with link agents to draw the lines between them.
Type 6. This draws the square using patch agents.

WHAT IS ITS PURPOSE?

The purpose is to illustrate how versatile NetLogo is at drawing. A secondary purpose is to show how turtle, patch and link agents in NetLogo are programmed, and to illustrate the use of the turtle drawing commands. A tertiary purpose is to illustrate how patterns can emerge through the execution of relatively straightforward commands. (If further interested, the reader should research the topics of L-systems and fractals).

HOW IT WORKS

The model uses a single turtle agent for Types 1 to 3, and draws the squares by manipulating the shape and size of the turtle to animate the squares growing larger, or by moving the turtle around the outer edge of the squares using either the turtle's pen or by stamping successive locations.

The model uses multiple turtle agents for Type 4, and once drawn, these turtles remain in the same place. i.e. There are many little square turtles that are used to draw the outside of the larger squares, like square Lego bricks all placed alongside each other.

The model uses link agents between four hidden turtle agents at the four corners for Type 5.

The model uses patch agents for Type 6. Like for the Type 4 turtle agents, each patch agent is used to draw the outside of the larger squares, like square Lego bricks all placed alongside each other.

HOW TO USE IT

Press the `setup` button first. This will clear the environment. Then choose the type of drawing you want using the `type-of-drawing` chooser. If you wish to draw the nested squares using six different methods, then set this chooser to `"Draw different squares"` or `"Types 1 to 6 repeated"`. Otherwise the squares will be drawn all the same using a single type as specified by the chooser.

If you wish the squares to be drawn from the inside out, then set the `draw-inside-out?` switch to `On`. The `innermost-size` slider value defines the size of the innermost square, and therefore the size of the hole at the centre. The `number-of-squares` slider value defines how many squares to draw when the `type-of-drawing` chooser is set to drawing the squares all the same (Types 1 to 6).

To draw the squares, press either the `draw all squares once` button, which will draw them once only, or if you wish them to be drawn continuously, then press the `draw all squares forever` button.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment.
- `draw all squares once`: This will draw all the nested squares once only.
- `draw all squares forever`: This will draw all the nested squares continuously.
- `draw-innermost-square`: This will draw the square in the centre, using the Type 1 method.
- `draw-inner-square`: This will draw the next square out from the centre, using the Type 2 method.
- `draw-middle-inner-square`: This will draw the next square out, using the Type 3 method.
- `draw-middle-outer-squa`re: This will draw the next square out, using the Type 4 method.
- `draw-outer-square`: This will draw the next square out, using the Type 5 method.
- `draw-outermost-square`: This will draw the outside square, using the Type 6 method.

The model's `Interface` sliders, chooser and switch are defined as follows:

- `type-of-drawing`: This defines the method used for drawing. The different types are defined as follows:
  `"Draw different squares"`: This will draw six different squares using different methods – defined as Type 1 to Type 6 (see above).
  `"Type 1 (draw the same squares)"`: This will draw the squares all the same using the Type 1 method.

`"Type 2 (draw the same squares)"`: This will draw the squares all the same using the Type 2 method.
`"Type 3 (draw the same squares)"`: This will draw the squares all the same using the Type 3 method.
`"Type 4 (draw the same squares)"`: This will draw the squares all the same using the Type 4 method.
`"Type 5 (draw the same squares)"`: This will draw the squares all the same using the Type 5 method.
`"Type 6 (draw the same squares)"`: This will draw the squares all the same using the Type 6 method.
`"Types 1 to 6 repeated"`: This will repeat drawing the nested squares, cycling through Types 1 to 6.

- `draw-inside-out?`: If this is set to `On`, the squares will be drawn from the inside out with the innermost square drawn first, otherwise they will be drawn from the outside in. (Try changing this as the squares are being drawn).
- `innermost-size`: This sets the size of the innermost square. As a consequence, it also defines the size of the inner hole.
- `number-of-squares`: This sets the number of squares to be drawn when the `type-of-drawing` is not set to `"Draw different squares"`.

THINGS TO NOTICE

Notice how the inner square (Type 2) has rounded corners. This is because it is drawn using the turtle's pen.

Notice how the middle inner square has very faint lines where the turtle squares have not overlapped with each other. To get rid of this, change `"set size 1"` to `"set size 1.1"` in the code when the first turtle is created.

Notice what happens when the `innermost-size` slider value is set to a large number, or the `number-of-squares-the-same` slider is set to a large number when the `type-of-drawing` chooser is not set to `"Draw different squares"`.

Notice which methods seem to take longer than the others at drawing the squares.

Notice that the Type 1 method (using a single turtle with shape `"square 2"`) only allows one square to be visible at any one time. Why is this? What happens when you choose Type 1 and press the `draw all squares forever` button?

Notice that the Type 2 (using a single agent's pen) and Type 5 methods (using four hidden turtle agents plus link agents between them) draw some of the sides of the squares in the wrong direction when the size of the side gets long (towards the edge of the world rather than towards the other corners). Why is this?

Notice that for Type 6, if you slow down the `Interface` speed slider, it will show the individual patches being drawn at random locations around the edge of each square.

THINGS TO TRY

Try slowing down the speed using the `Interface`'s speed slider. You will be able to see how the different squares are drawn.

Try running this model in 3D mode for the various configurations. What happens and why?

NETLOGO FEATURES

This demonstrates how to draw using three different types of agents – turtle agents, link agents and patch agents.

EXTENDING THE MODEL

Are there further ways for drawing the squares?

RELATED MODELS

See the Nested Triangles model.

Solution to Exercise 3.6.9:

The ANZ Continental Drift model illustrates a partial solution to this exercise. See Solution to Exercise 6.3.2 in the book '*Exercises for Artificial Intelligence – Agent Behaviour I*'.

Solution to Exercise 3.6.10: Shuffle Cards NetLogo Model and Shuffle and Deal Cards NetLogo Model

The following NetLogo model provides a solution to this exercise:

Shuffle Cards	http://files.bookboon.com/ai/Shuffle-Cards.html

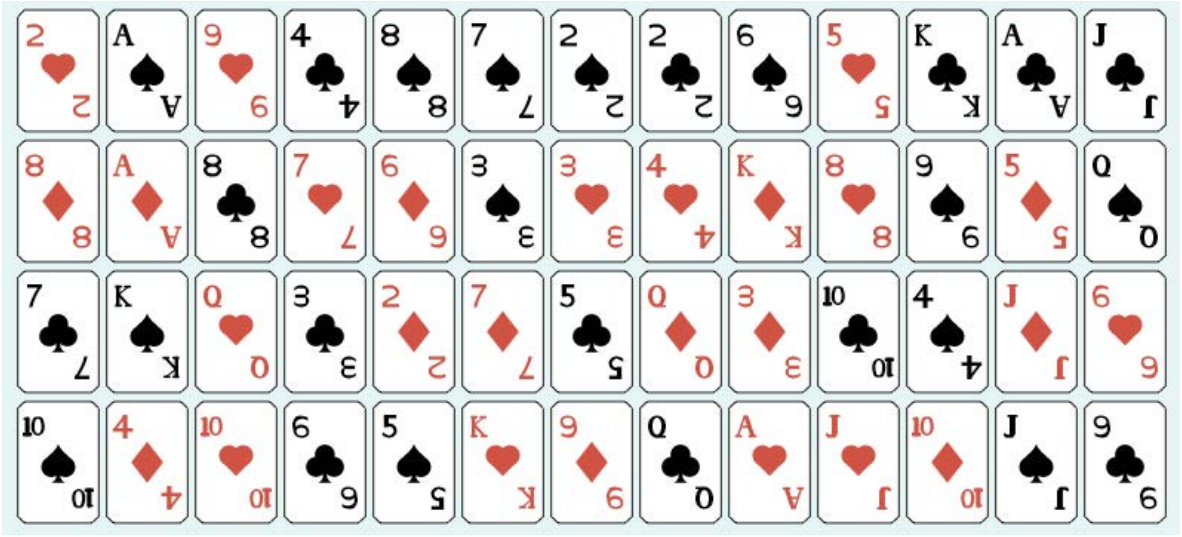


Figure S-3.6.10.1 Screenshot of the Shuffle Cards model.

WHAT IS IT?

This model shows how you can create a pack of cards using turtle shapes and then shuffle them using the `shuffle` command.

WHAT IS ITS PURPOSE?

Its purpose is to illustrate that a lot can be done using just turtle shapes.

HOW IT WORKS

The model uses three breeds of turtle agents to represent each card in the pack. These are: `cards`, `suits` and `ranks`. Each of these agents has a particular shape which is used to visually depict a card in the pack when it is shown. The `card` turtle agent owns a suit (e.g. clubs, spades, diamonds and hearts; this is represented by a `suit` turtle agent) and two ranks (e.g. Ace, King, represented by two `rank` turtle agents). These ranks appear at the top and bottom of a card – each of these is the 180 degrees rotation of the other.

Each card in the pack is created as separate agents (`card`, `card-suit`, `card-rank-1` and `card-rank-2`) and this is then stored in a global list called the `pack`.

HOW TO USE IT

Select the slider values you want, then press the `Shuffle` button to display a shuffled deck of cards.

THE INTERFACE

The model's `Interface` button and sliders are defined as follows:

- `Shuffle`: This button clears the environment, creates a new pack, then shuffles it, displaying the result in the environment.
- `card-size`: This is the size of each card.
- `suit-size`: This is the size of each suit depicted in the middle of each card.
- `rank-size`: This is the size of each rank depicted to the top left and bottom right of each card.

THINGS TO NOTICE

Notice the extra shapes that have been added to the shapes library for the model by running the `Turtles Shapes Editor` in the `Tools` menu of the `Interface`. You will find a number of shapes added – these begin with `"number: …"` and `"suit: …"`.

Notice how the card shape has been drawn using the shapes editor. Click on the outside of the card to verify that the thin outside line exists in the shape specification.

Notice the order that the agent positions are specified using the `setxy` command in the `shuffle-pack-of-cards` procedure for the three type of agents that are used to depict each card. The order is the `card` agent drawn first, followed by the `card-suit`, `card-rank-1` and `card-rank-2` agents. This ensures that the `card` agent appears underneath the other three agents and does not obscure them.

THINGS TO TRY

Try changing the values of the sliders to see what happens.

Try commenting out the last line of the setup procedure (`set pack shuffle-pack-of-cards pack -90 30`) to see what happens.

Have a go at improving the letter and number shapes that are used to specify the card rank.

EXTENDING THE MODEL

Create a model that can be used to play a card game. One trick that you can use is to hide the cards in the environment (using the `hide-turtle` command) and only show them when required. (See the Shuffle and Deal model to see how this could be done).

NETLOGO FEATURES

The `shuffle` command in NetLogo makes shuffling the pack of cards very easy.

RELATED MODELS

See the Shuffle and Deal model.

The following NetLogo model extends this model by adding the option of dealing the pack of cards once it has been shuffled:

Shuffle and Deal Cards	http://files.bookboon.com/ai/Shuffle-and-Deal-Cards.html

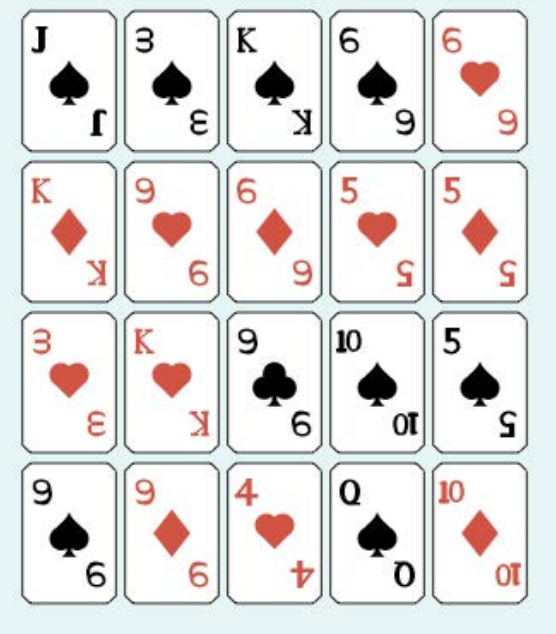


Figure S-3.6.10.2 Screenshot of the Shuffle and Deal Cards model when the number-of-hands slider is 4 and the cards-in-the-hand slider is 5.

This `Information` for this model is the same as for the Shuffle Cards model above except for the following.

WHAT IS IT?

This model is a modification of the Shuffle Cards model. It shows how you can create a pack of cards using turtle shapes, shuffle them using the `shuffle` command and then deal them out.

WHAT IS ITS PURPOSE?

Its purpose is to illustrate that a lot can be done using just turtle shapes. A secondary purpose is to demonstrate how information can be stored in the environment using the `hide-turtle` command and then displayed when required using the `show-turtle` command.

HOW IT WORKS

The cards are "dealt" by first hiding all the cards in the pack, then moving the agents to the required locations.

HOW TO USE IT

Select the slider values you want, then press the `Shuffle` button to display a shuffled deck of cards. The `Deal` button will then deal out the cards. If you then subsequently press the `View pack` button, this will show how the cards that have been dealt from the pack have been moved and are now overlaying some of the remaining cards in the pack.

THE INTERFACE

The extra buttons and sliders added to the model's `Interface` is defined as follows:

- `Deal`: This button will 'deal' out the cards into a number of hands as determined by the `number-of-hands` and `cards-in-the-hand` sliders.
- `Sort hands 1`: This button will sort the cards in the hand by suit then type order.
- `Sort hands 2`: This button will sort the cards in the hand by type then suit order.
- `View pack`: This button will show where all of the agents are currently located in the environment (it uses the `show-turtle` command to show all the cards). If the cards in the pack have already been dealt out, some will have been moved and therefore overlay other cards in the pack.
- `number-of-hands`: This is the number of hands that the cards will be dealt to.
- `cards-in-the-hand`: This is the number of cards in the hand.

THINGS TO NOTICE

When the `View pack` button is pressed after the cards have been dealt, notice the pattern of which cards now overlay each other and which cards have been moved (i.e. there are blank spaces). Which bit of code causes this?

THINGS TO TRY

Try changing the model so that when you press the `"View the pack"` command it will move any cards that have been dealt back to their original locations in the shuffled pack.

NETLOGO FEATURES

The `hide-turtle` and `show-turtle` commands are also a powerful way of managing program state.

RELATED MODELS

See the Shuffle model.

Solution to Exercise 3.6.11: Sudoku Builder NetLogo Model

The following NetLogo model provides a solution to this exercise:

Sudoku Builder	http://files.bookboon.com/ai/Sudoku-Builder.html

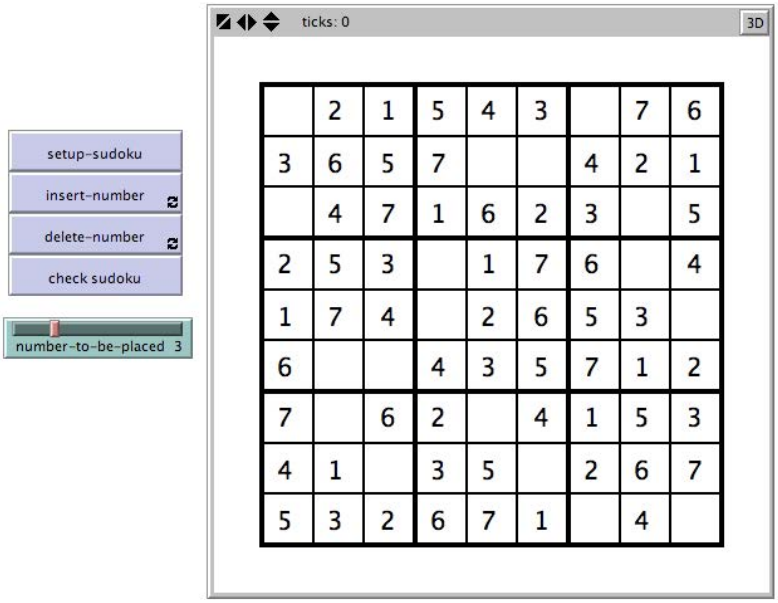


Figure S-3.6.11 Screenshot of the Sudoku Builder model after the setup-sudoku button has been pressed and then the shown numbers inserted using the insert-number button.

WHAT IS IT?

This model allows the user to fill in a Sudoku puzzle.

WHAT IS ITS PURPOSE?

Its purpose is to demonstrate the use of various NetLogo commands and to show how relatively easy it is to create a non-trivial visual user interface that would require much more work in other languages.

HOW IT WORKS

It does this by creating a separate turtle agent (using a breed called a `'square'`) for each cell of the 9 by 9 Sudoku grid. Each square turtle agent keeps a `number` variable for storing the number that has been placed in the cell. Each agent also has two further variables – `square-x` and `square-y` – which record the (*x*, *y*) position of the square in the grid, where the bottom left square is (1, 1) and the top right is (9, 9). This is used to make it easier for checking the correctness of the placed numbers.

HOW TO USE IT

To set up an empty Sudoku grid, press the `setup-sudoku` button. To insert a number as specified by the `number-to-be-placed` slider into the grid, press the `insert-number` button. To delete a number, press the `delete-number` button. Pressing the `check-sudoku` button will check that the placed numbers are placed correctly.

THE INTERFACE

The model's `Interface` buttons and slider are defined as follows:

- `setup-sudoku`: This will create an empty Sudoku puzzle without any numbers inside it.
- `insert-number`: This will insert a number into a cell of the Sudoku puzzle
- `delete-number`: This will delete a number from a cell of the Sudoku puzzle.
- `check-sudoku`: This will check whether the placement of numbers in the Sudoku grid is correct or not. That is, there are no duplicate numbers in any horizontal or vertical line, or any 3 by 3 grid.
- `number-to-be-placed`: This is the number that will be placed when the mouse is clicked after the `insert-number` button is pressed.

THINGS TO NOTICE

This model provides the user with the ability to create a Sudoku puzzle for themselves, or try to fill in an existing one. Hence, there is not a lot to notice, except to say that if you use this model to try to create your own Sudoku puzzle, you will find it quite difficult. As the grid becomes filled in, it becomes increasingly more difficult to find a correct placement. Often you will find that proceeding further with the current configuration will be impossible, and you will need to backtrack to a point where a solution is still possible. Usually starting again can be the easiest option in this case.

THINGS TO TRY

Try using the model to solve an existing Sudoku puzzle.

Also, try creating your own Sudoku puzzle.

EXTENDING THE MODEL

Try adding some code that will report whether a current configuration has a solution (i.e. the remaining numbers can still be placed correctly).

Try extending the model so that it will automatically create a Sudoku puzzle. The hard bit is to find a configuration that makes it possible for a human to solve it using different levels of difficulty (as occurs with standard puzzles provided in puzzle books, newspapers, and online).

NETLOGO FEATURES

Note the use of the `remove-duplicates` command in the `check-sudoku` procedure that makes it easy to check whether a horizontal line, vertical line or 3 by 3 grid has any duplicate numbers in them (which indicates that the placement is incorrect).

Solution to Exercise 3.6.12:

See the Solution to Exercise 5.5.1.

Solution to Exercise 3.7.3:

As an example of what is possible, look at the Butterfly Maze in the Mazes-2 NetLogo model described in the solution for Exercise 5.4.1 below.

Solution to Exercise 3.7.4: Chevening House Maze with Coloured Islands NetLogo Model

The following NetLogo model provides a solution to this exercise:

Chevening House Maze with Coloured Islands	http://files.bookboon.com/ai/Chevening-House-Maze-with-Coloured-Islands.html

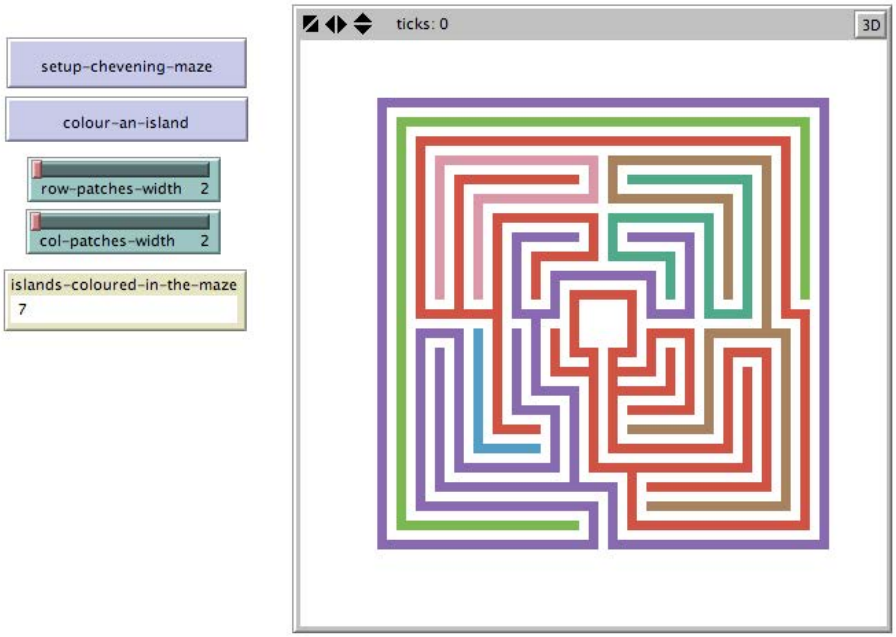


Figure S-3.7.4. Screenshot of the Interface for the Chevening House Maze with Coloured Islands model after the setup-chevening-maze button has been pressed followed by the colour-an-island button seven times.

The `Information` for this model is the same as for the Chevening House Maze model, except for the following:

WHAT IS IT?

This model is a modification of the Chevening House Maze model where the user can incrementally colour the islands in the maze.

THE INTERFACE

The `setup-chevening-maze` button will redraw the maze.

The `colour-an-island` button will colour one of the islands in the maze that has not already been coloured.

The `islands-coloured-in-the-maze` monitor keeps a running total of the number of islands that have been coloured so far. This number is incremented each time the `colour-an-island` button is pressed.

HOW IT WORKS

The colouring algorithm uses a recursive procedure to fan out from a single initial blue patch selected at random. It does this by colouring all the patch's neighbours, then all the neighbours' neighbours, and so on, until there are no more neighbours that are still blue.

HOW TO USE IT

To have the model draw the maze, press the `setup-chevening-maze` button.

To colour one of the islands in the maze, press the `colour-an-island` maze. Keep on pressing this button until all the islands have been coloured. How many islands were there?

WHAT IS ITS PURPOSE?

Its purpose is to show that the Chevening House maze is a multiply connected maze i.e. that the centre of the maze cannot be reached using the hand-on-the-wall behaviour since the centre is within an island.

THINGS TO TRY

Try slowing down the speed during the colouring of each island. Some of the patches, often the ones at right-angled corners, remain uncoloured for a while but get coloured latter on. Why does this happen?

EXTENDING THE MODEL

How efficient is the colouring algorithm? Is there a more efficient way to perform the colouring?

Solution to Exercise 4.2.1: Simple Walk NetLogo Model

```
left 90
forward 28
right 90
forward 56
right 90
forward 56
right 90
```

The Simple-Walk NetLogo model illustrates the solution to this exercise. Try it out in NetLogo:

Simple Walk	http://files.bookboon.com/ai/Simple-Walk.html

WHAT IS IT?

This model illustrates solutions to Exercises 4.2.1 and 4.2.2.

HOW TO USE IT

Press the `setup-maze` button first, then press the `setup-turtle` button. The `walk` button will execute the solution for Exercise 4.2.1. The `shortest-walk-1`, `shortest-walk-2` and `shortest-walk-3` buttons will execute solutions for Exercise 4.2.2.

Solution to Exercise 4.2.2:

Solutions to this question are illustrated by Simple-Walk NetLogo model. Try it out in NetLogo using the link provided in the solution for Exercise 4.2.1 above.

One solution (implemented by the `shortest-walk-1` procedure in the model) is to assume only vertical and horizontal movement is allowed (based on the answer to Exercise 4.2.1). Then the solution is:

```
forward 56
right 90
forward 28
right 90
```

The following solution (implemented by the `shortest-walk-2` procedure in the model) takes just two steps:

```
setxy 30 30
right some-angle
; work out the value of "some-angle" for yourself
```

Another solution (implemented by the `shortest-walk-3` procedure in the model) uses one further step:

```
facexy 30 30
forward sqrt (56 * 56 + 28 * 28)
; (using square of the hypotenuse)
right (180 - 26.6)
; (using angle calculation for a right angle triangle)
```

Solutions to Exercises 4.2.3, 4.2.4 and 4.2.5: Nested Triangles NetLogo Model

The Nested-Triangles NetLogo model illustrates the solution to this exercise. Try it out in NetLogo:

Nested Triangles	http://files.bookboon.com/ai/Nested-Triangles.html

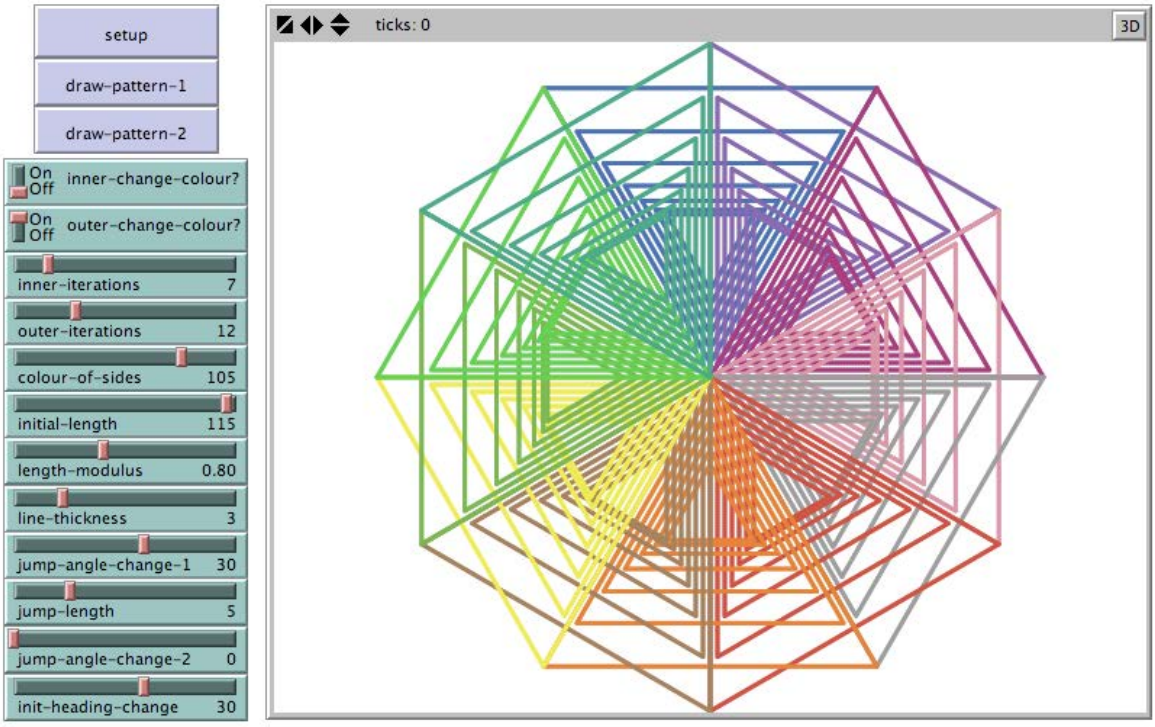


Figure S-4.2.3-1. Screenshot of the Interface for the Nested Triangles model after the setup button has been pressed followed by the draw-pattern-1 button, with the chooser and slider values as shown in the image.

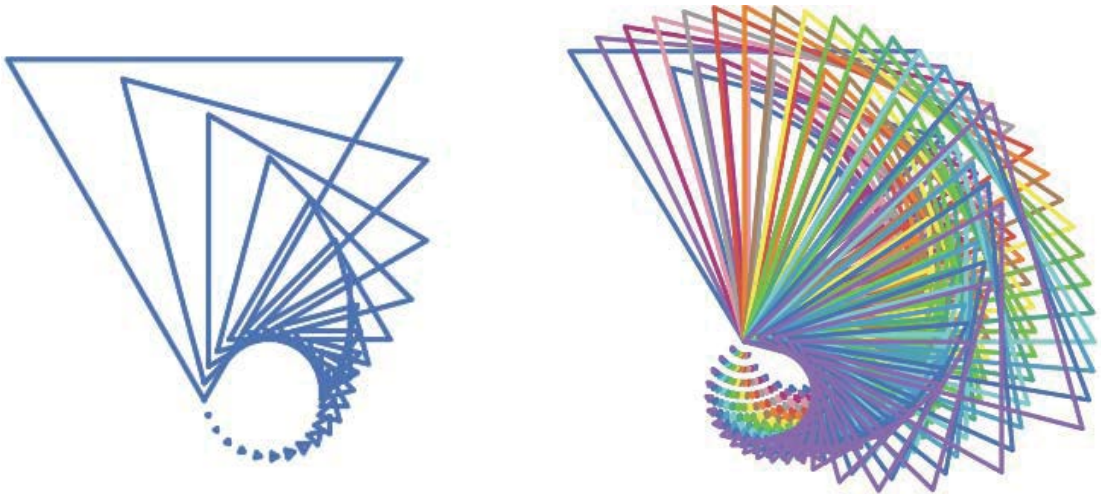


Figure S-4.2.3-2. Screenshots of patterns produced by the Nested Triangles. The left image was produced by pressing the draw-pattern-1 button, the right image by pressing the draw-pattern-2 button. The values of the sliders for these images are the same as shown in Figure S-4.2.3-1, except for the following: inner-iterations = 40; outer-iterations = 60; jump-angle-change-2 = 15; init-heading-change = 5.

WHAT IS IT?

This model illustrates solutions to Exercises 4.2.3, 4.2.4 and 4.2.5 for the *Exercises for Artificial Intelligence* book series. It also shows what you can do with a single turtle agent wandering around drawing things.

WHAT IS ITS PURPOSE?

Its purpose is to show how turtle agents in NetLogo are programmed, and to illustrate the use of the turtle drawing commands. A secondary purpose is to illustrate how patterns can emerge through the execution of relatively straightforward commands. (If further interested, the reader should research the topics of L-systems and fractals).

HOW IT WORKS

The model uses a turtle agent to draw a set of nested equilateral triangles. The outer triangle is drawn first, then the pen is pulled up while the turtle jumps to the next inner triangle using a jump angle and length specified by slider values. The `length-modulus` is a ratio that is used to reduce the size of each successive triangle. The `inner-iterations` slider is used to specify the number of triangles drawn.

The value of the `outer-iterations` slider controls the number of times the set of triangles are redrawn. Once all the triangles in one particular set are drawn, then the turtle changes it heading by adding the `init-heading-change` slider value to its current heading. For example, if this value is 30, and the value of the `outer-iterations` slider is set to 12, then the pattern will cover the full 360 degrees (if the turtle returns to its origin on its same heading once each set of triangles is drawn).

HOW TO USE IT

Press the `setup` button first, then press either the `draw-pattern-1` or `draw-pattern-2` button.

The `draw-pattern-1` button can be used to produce the patterns for Exercises 4.2.3 and 4.2.4. (For the values of the switches and sliders used to produce the pattern in Exercise 4.2.3, see Figure S-4.2.3-1 but change the value of the `inner-iterations` slider to 20. For Exercise 4.2.4, use the same switch and slider values, but change the `inner-change-colour?` switch to `On`, and change the `jump-angle-change-2` slider value to 15.)

The `draw-pattern-2` button can be used to produce the pattern shown for Exercise 4.2.5 and also shown in Figure S-4.2.3-1. (For the former, use the same values as shown for the switches and sliders in Figure S-4.2.3-1 but set the `outer-change-colour?` switch to `Off`).

The settings for two further patterns are described in Figure S-4.2.3-2.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This will clear the environment, reset all variables and set all patches to white.
- `draw-pattern-1`: This draws a single set of (usually nested) triangles. The number of triangles drawn is determined by the value of the `inner-iterations` slider.
- `draw-pattern-2`: This draws a multiple set of (usually nested) triangles. The number of sets drawn is determined by the value of the `outer-iterations` slider.

The model's `Interface` switches and sliders are defined as follows:

- `inner-change-colour?`: This switch if set to `On` will change the colours of the triangles as they are drawn successively.
- `outer-change-colour?`: This switch if set to `On` will change the drawing colour at the start of a new set of triangles.
- `inner-iterations`: This is the number of (usually nested) triangles that are drawn in each set of triangles.
- `outer-iterations`: This determines the number of sets that are drawn.
- `colour-of-sides`: This is the initial colour that is used to draw the sides of each triangle.
- `initial-length`: This is the initial length of the sides of the outer triangle in each set.
- `length-modulus`: This ratio controls how much the length of the sides of the triangle is reduced or enlarged each time a new triangle is drawn.
- `line-thickness`: This is the thickness of the lines that are drawn by the turtle (i.e. this is the width of the sides of the triangles).

- `jump-angle-change-1`: This specifies the change of direction in degrees when the turtle jumps across to draw the next triangle.
- `jump-length`: This specifies the distance the turtle moves (with its pen up temporarily) when it jumps across to start drawing the next triangle.
- `jump-angle-change-2`: This specifies the value of a further directional change the turtle makes after each jump.
- `init-heading-change`: This is the change of direction in degrees made by the turtle when it has completed drawing a set of triangles. This is done before the start of drawing the next set of triangles.

THINGS TO NOTICE

Notice the infinite variety of patterns that can be created using a set of simple commands. Notice also that although the sets of triangles are drawn locally (i.e. using a turtle executing a small set of commands from a first-person perspective based on its current location), global patterns emerge from the interactions between the different sets.

THINGS TO TRY

Try changing the values of the switches and sliders to create further patterns.

Try changing the values of the sliders dynamically as the `draw-pattern-1` and `draw-pattern-2` buttons are still pressed. For example, change the jump angles and length, or change the colour.

EXTENDING THE MODEL

Change the model so that other shapes can be drawn – for example, isosceles triangles, squares, rectangles, hexagons etc.

Change the model so that the user can specify: 1) the set of commands to be executed when the object (triangle, square, hexagon etc.) is drawn; 2) the set of commands to be executed when the jump is made; and 3) the set of commands to be executed when the drawing of a set of objects is completed before the start of the drawing of the next set of objects. This will give the user greater control over how the patterns are drawn, and greatly increase the variety of patterns that are possible.

NETLOGO FEATURES

The model demonstrates the use of turtle drawing commands.

Solution to Exercise 4.2.6: Hampton Court Maze with Turtle NetLogo Model

The Hampton Court Maze with Turtle model provides a solution to the problem of how get a turtle to move around the Hampton Court Maze. Try it out in NetLogo:

Hampton Court Maze with Turtle	http://files.bookboon.com/ai/Hampton-Court-Maze-with-Turtle.html

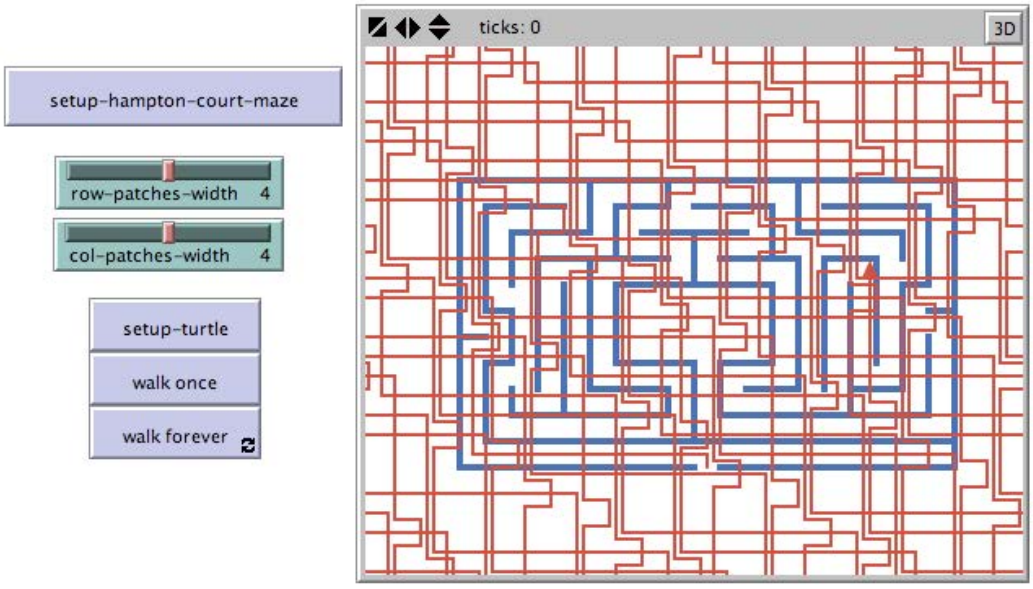


Figure S-4.2.6. Screenshot of the Interface for the Hampton Court Maze with Turtle model after the setup-hampton-court-maze button has been pressed followed by the setup-turtle and the walk forever buttons, with the slider values as shown in the image, when the last 8 commands in the walk procedure have been commented out.

WHAT IS IT?

This model gets a turtle to wander through the beginning of the Hampton Court maze as created by the Hampton Court NetLogo model.

HOW IT WORKS

This adds a simple turtle agent to the Hampton Court Maze model. The turtle executes explicit commands – `rt`, `lt` and `fd`. The turtle is initially placed at the entrance to the maze using a `setxy` command. These explicit commands will only work on the Hampton Court Maze, and only when the two sliders, `row-patches-width` and `col-patches-width`, are set to 4.

WHAT IS ITS PURPOSE?

The purpose is to show how the simple turtle commands for movement – `rt`, `lt` and `fd` – work from a first person perspective. The only way for a programmer to get the commands for a turtle to work properly in a maze type environment is to mentally imagine being in the same situation as the turtle itself and then have it turn left or right, or head forward depending on its current location. The model also shows how the turtle-drawing command, `pen-down`, works.

A secondary purpose is to show that explicitly programming a solution to a problem is of limited use – it cannot be applied in any other situation. For a more general solution, the turtle has to be programmed with an ability to sense what is in the environment and then respond to what it senses. How it does this determines its behaviour and how successful this behaviour is at solving the problem.

HOW TO USE IT?

Set both slider values to 4 first, and press the `setup-hampton-court-maze` button to set up the maze. Then press the `setup-turtle` button followed by the `walk` button.

THE INTERFACE

The `Interface` buttons and sliders are defined as follows:

- `setup-hampton-court-maze`: This clears the environment and draws the Hampton Court Maze.
- `row-patches-width`, `col-patches-width`: This sets up the dimensions of the rows and columns. Setting these values to anything but 4 will mean the turtle will start in the wrong place (not at the entrance) and will walk in the wrong direction (through walls etc.)
- `setup-turtle`: This creates a single turtle and positions it at the entrance of the maze if the two slider values are set at 4.
- `walk once`: This will make the turtle walk the first part of the maze if the two slider values are set at 4.
- `walk forever`: This continuously repeats the same instructions for walking the first part of the maze.

THINGS TO TRY

The turtle will only walk correctly when the two sliders, `row-patches-width` and `col-patches-width`, have been set to 4. Try setting the slider values to something else. What happens and why?

Try pressing the `walk` button multiple times to see what happens. Then try commenting out the last three commands from the `walk` procedure. Then again repeat pressing the `walk` button to see what happens. Next comment out the last six commands instead of three and repeat the multiple walks. Finally, try commenting out the last eight commands and repeat the multiple walks. Under what conditions does the turtle repeatedly return to the entrance of the maze?

EXTENDING THE MODEL

Try extending the model so that it will work regardless of the values in the two sliders, `row-patches-width` and `col-patches-width`.

Try defining the behaviour of the turtle so that it can sense the maze and react accordingly in order to walk through the maze without having to explicitly define every single movement command. (For some possible solutions, see the Mazes and Searching Mazes models).

NETLOGO FEATURES

This model provides an illustration of how the `rt`, `lt` and `fd` commands work from a first-person perspective. It also shows how the `pen-down` command can be used to draw a trace of a turtle's path.

RELATED MODELS

See the Hampton Court Maze model, the Hampton Court Maze with Wall Following model, the Mazes model and the Searching Mazes model.

Solution to Exercise 4.2.7: Foxes and Rabbits 2 NetLogo Model

The Foxes and Rabbits 2 model provides a solution to this problem. Try it out in NetLogo:

Foxes and Rabbits 2	http://files.bookboon.com/ai/Foxes-and-Rabbits-2.html

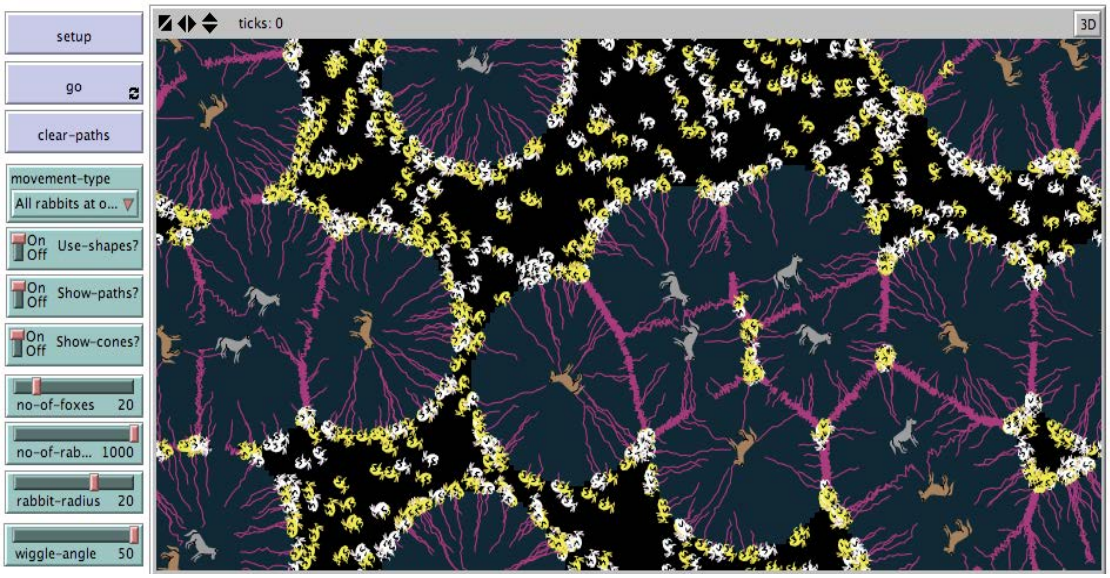


Figure S-4.2.7.1. Screenshot of the Interface for the Foxes and Rabbits 2 model after the setup button has been pressed followed by the go button, with the slider values as shown in the image.

WHAT IS IT?

This model shows how to use breeds to create two different breeds of turtle agents – foxes and rabbits – that can have different properties such as colour. Once the foxes and rabbits have been created, the rabbits that are too near to the foxes move away until they feel they are at a safe distance. The colour of the rabbit or rabbits is temporarily switched to magenta when they are moving away.

HOW TO USE IT

Press the `setup` button several times to see how the agents spread themselves throughout the environment. Then press the `go` button to see the rabbits move away from the foxes.

To show the paths of the rabbits, turn the `Show-paths?` switch to `On`. To clear the paths, press the `clear-paths` button.

If there are many agents, you will need to speed up the simulation using the speed slider in the `Interface`.

THE INTERFACE

The `Interface` buttons are defined as follows:

- `setup`: This will create a new population of foxes and rabbits at random locations.
- `go`: This will start the simulation. All the rabbits will move away from the foxes.
- `clear-paths`: This clears the rabbits' paths if any are drawn.

The `Interface` chooser, switches and sliders are defined as follows:

- `movement-type`: This controls the behaviour of the rabbits. If set to `"All rabbits at once"`, the rabbits move apart all at the same time, one step at a time. If set to `"One rabbit at a time"`, the rabbits will take turns until each is at a satisfactory distance from the fox nearest to it. (This is admittedly rather unrealistic behaviour, a bit like what happens in Kung Fu movies when the baddies attack the hero, but do this one at a time).
- `Use-shapes?`: If set to `On`, this will draw the agents using fox and rabbit shapes. If set to `Off`, it will fall back to the default shape.
- `Show-paths?`: If set to `On`, this will draw the paths of the rabbits that are moving.
- `Show-cones?`: If set to `On`, this will draw a cone around the foxes showing the distances within which the rabbits will feel that it is not safe. (This distance is controlled by the `rabbit-radius` slider).
- `no-of-foxes`: This is the number of foxes that will be created when the `setup` button is pressed.

- `no-of-rabbits`: This is the number of rabbits that will be created when the `setup` button is pressed.
- `rabbit-radius`: This is the distance away from a fox for which the rabbit will feel unsafe.
- `wiggle-angle`: This controls the wiggle behaviour of the rabbit as it moves. A larger number results in more random movement; a value of 0 will result in the rabbit heading in straight lines unless it comes too near another fox.

HOW IT WORKS

It uses two turtle agent breeds, foxes and rabbits, and shows how you can use the `turtles-own` command so that both breeds end up with common variables – `age` and `gender`.

The `setup` command simply calls the `create-foxes` and `create-rabbits` command to create the agents. These are created at random locations. The foxes move apart slightly to avoid being too near each other. The rabbits will turn away from the nearest fox.

The `go` command directs the rabbits that are nearby to each fox to move away all at the same time or one at a time, depending on the value of the `movement-type` slider.

WHAT IS ITS PURPOSE?

Its purpose is to show how to define and create breeds of agents, and provide an example of agent movement.

THINGS TO NOTICE

Notice the different types of behaviour that arise when the movement-type chooser is set to `"All rabbits at once"` and `"One rabbit at a time"`. Even though the base rabbit behaviour is the same for both – that is, both use the same `run-away` procedure – the overall system behaviour is very different. When the movement-type slider is set at `"All rabbits at once"`, quite a few of the rabbits end up being "trapped", usually between three equidistant foxes. Also, if you turn on the `Show-paths?` switch, the paths that are drawn are very different for the different types of movement.

THINGS TO TRY

Try changing the values of the chooser, switches and sliders to see what happens.

Try reproducing similar screenshots to that shown in Figure S-4.2.7.2 below. For the first one, the number of foxes has been set at 20 and the number of rabbits at 1000. For the second one, the number of foxes has been set at 40, and the number of rabbits at 1000.

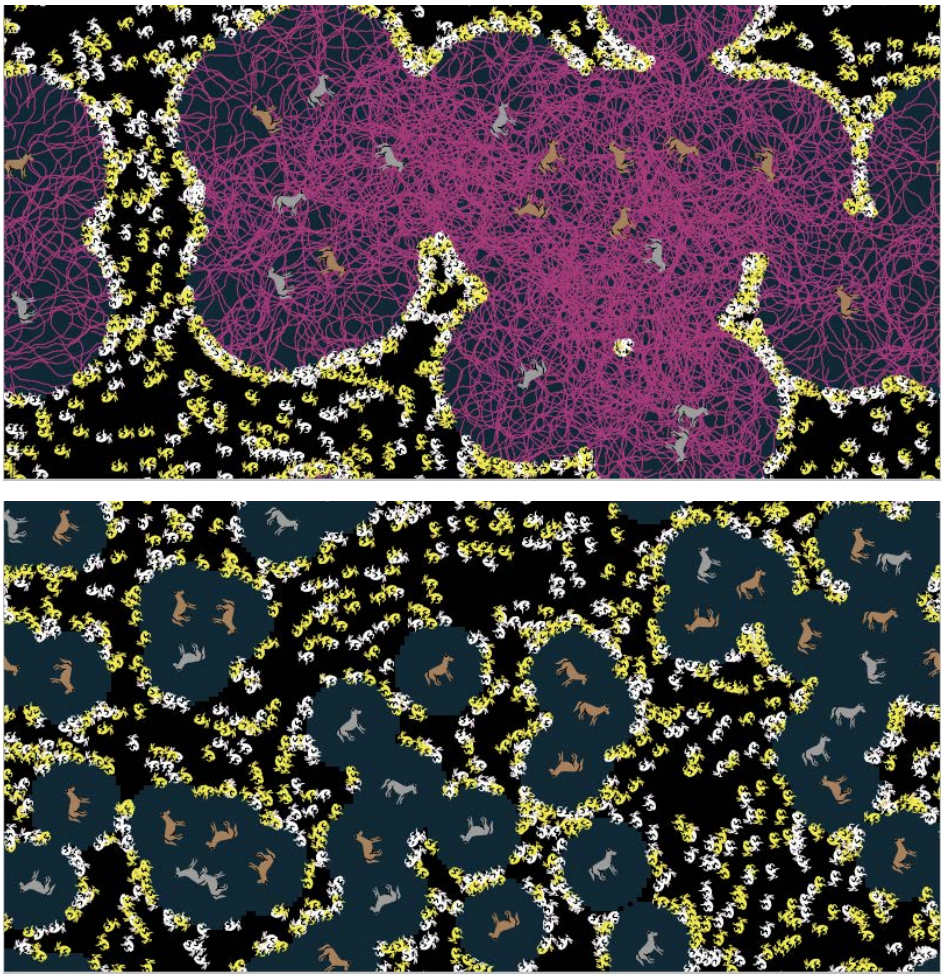


Figure S-4.2.7.2. Further screenshots for the Foxes and Rabbits 2 model.

EXTENDING THE MODEL

Try having the foxes move around chasing the rabbits rather than stay still.

RELATED MODELS

See the Foxes and Rabbits model.

Solution to Exercise 4.3.1:

See the Solution to Exercise 5.5.1.

Solution to Exercise 4.5.5: Agent Animation Model

The Agent Animation model provides a solution to the problem of how to animate the movement of agents with different shapes across the environment from left to right. Try it out in NetLogo:

Agent Animation	http://files.bookboon.com/ai/Agent-Animation.html

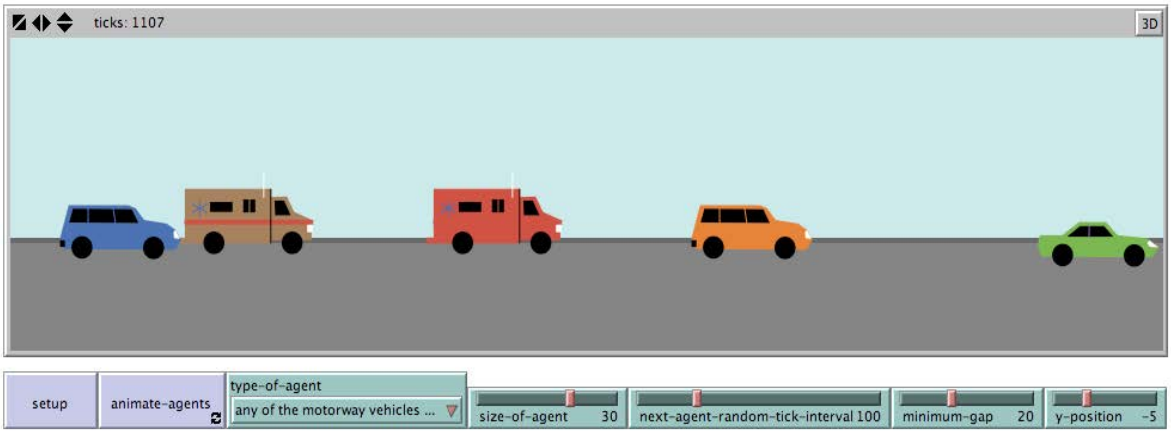


Figure S-4.5.5. Screenshot of the Interface for the Agent Animation model after the setup button has been pressed followed by the animate-agents button, with the slider values as shown in the image.

WHAT IS IT?

This model performs a simple animation of various turtle agent shapes to give the impression that they are flowing past the observer. The user can select the size and shape of the agent, its position on the *y*-axis, the random interval at which the next agent will appear and the minimum gap between agents.

WHAT IS ITS PURPOSE?

The purpose of this model is to show how a simple translation movement can be used to perform basic animation of turtle agents.

This model illustrates a solution to Exercise 4.5.5 for the book "Exercises for Artificial Intelligence – Agents and Environments".

HOW IT WORKS

Turtle agents are created at random intervals on the left side of the environment. These are then moved horizontally across the environment by incrementing their *x* co-ordinate.

HOW TO USE IT

To run the animation, press the `setup` button followed by the `animate-agents` button. Changing the values of the chooser and sliders in the `Interface` during the animation will change the nature and frequency of the agents that appear in the animation.

THE INTERFACE

The model's `Interface` buttons are defined as follows:

- `setup`: This resets the animation and draws the background.
- `animate-agents`: This starts the animation in the environment.
- The model's `Interface` chooser and sliders are defined as follows:
- `type-of-agent`: This chooser selects the shape of the agent that appears next.
- `size-of-agent`: This is the size of the agent that appears next.
- `next-agent-random-tick-interval`: This controls how often the agents appear. The next agent will appear at an interval after the previous agent which is a random number from 0 up to this number + the value of the `minimum-gap` slider.
- `minimum-gap`: This is the minimum gap with which the agents appear.
- `y-position`: This is the *y* position of the agent which remains unchanged as the agent is moved across from left to right.

THINGS TO NOTICE

Notice which of the shapes (as defined by the `type-of-agent` chooser) seem to create realistic animation and which do not. The animation of the person, cow, mouse and sheep agents, for example, seem strange, and look like they are moving on a conveyor belt. Why?

THINGS TO TRY

Try changing the values of the chooser and sliders as the animation is running.

Try setting the value of the `next-agent-random-tick-interval` slider to 0 and the value of the `minimum-gap` slider to 1.

EXTENDING THE MODEL

Try making the motorway animation more realistic i.e. when the `type-of-agent` chooser is set to `"any of the motorway vehicles below"`. You will need to vary the size of some of the vehicle agents so that they end up similar sizes (for example, the `"car"` shape is too big in comparison to the other shapes). You will also need to vary the *y* positions of the different vehicle agents.

Try modifying the railway animation so that longer trains of agents appear rather than a single agent one at a time.

RELATED MODELS

See the Cars Guessing Game model, and the Stick Figure Walking and Stick Figure Animation models.

Solution to Exercise 5.3.1: Hill Climbing with Wall Following NetLogo Model

The Hill Climbing with Wall Following model provides one solution to the problem of how to combine multiple sensing behaviours. Try it out in NetLogo:

Hill Climbing with Wall Following	http://files.bookboon.com/ai/Hill-Climbing-with-Wall-Following.html

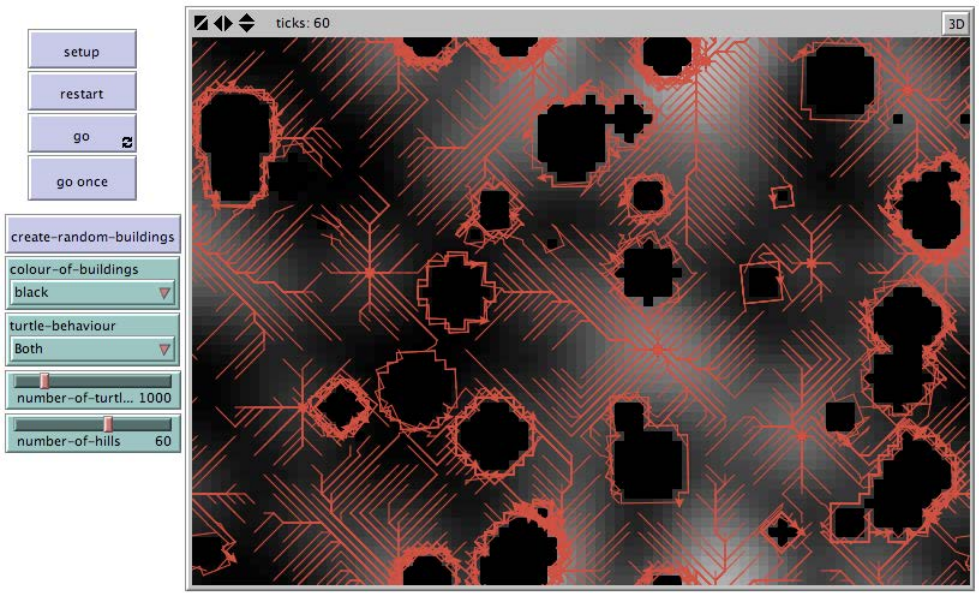


Figure S-5.3.1. Screenshot of the Interface for the Hill Climbing with Wall Following model after the setup button has been pressed followed by the go button, with the chooser and slider values as shown in the image.

WHAT IS IT?

Imagine blind turtles in mountainous terrain who want to find the nearest high points. Their situation is complicated by the presence of buildings in the terrain. The blind turtles only have two senses they can use – an ability of touch (i.e. they can follow a wall via proximity detection), and an ability to sense which way is up in the terrain. This model tries to simulate where the blind turtles would head to.

The model is based on two models provided by Uri Wilensky – the Hill Climbing Example model and the Wall Following Example model, both available in the Code Examples directory of the Models Library.

WHAT IS ITS PURPOSE?

The purpose of this model is to investigate what would happen when the two behaviours of hill climbing and wall following are combined. This is also a partial answer to Exercise 5.3.1.

HOW TO USE IT

First press the `setup` button after choosing the turtle behaviour you want to be exhibited using the `turtle-behaviour` chooser and selecting the number of turtles you want created using the `number-of-turtles` slider. To run the model, press either the `go` button for continuous simulation, or the `go once` button to run the simulation for a single tick. To add some buildings into the environment, press the `create-new-buildings` button. The `colour-of-buildings` slider sets the colour of the buildings. The number of hills (represented by light shading) is set by the `number-of-hills` slider.

THE INTERFACE

The buttons in the model's `Interface` are defined as follows:

- `setup`: This resets the simulation. All the turtles are re-created, all existing buildings are deleted, and a new random terrain with new buildings is generated in the environment.
- `restart`: This recreates the turtles and deletes any existing paths, but keeps the existing terrain and buildings.
- `go`: This will run the simulation continuously.
- `go once`: This will run the simulation one step at a time.
- `create-random-buildings`: This will create some more random buildings in the environment. Be careful not to press this too much, otherwise the buildings will then cover the entire environment.

The choosers and sliders in the model's `Interface` are defined as follows:

- `colour-of-buildings`: This allows the use to choose the colour of the buildings – either black or white.
- `turtle-behaviour`: This allows the user to set the behaviour of the turtles. Try changing this when the simulation is running and see what happens.
- `number-of-turtles`: This sets the number of turtles that will be created when either the `setup` or `restart` buttons are next pressed.
- `number-of-hills`: This sets the number of hills in the terrain that will be created when the `setup` button is next pressed.

THINGS TO NOTICE

Notice when the turtle behaviour is set to `Both` that many of the turtles end up following walls, while the rest climb to the nearest high point. Why is this? Can you add two modifications of the `Both` behaviour where the turtles still keep on applying both types of behaviour initially but either all end up at high points or all end up following walls?

Notice what happens when the colour of the buildings is set to `white` and the turtle behaviour is set to `Hill Climbing`. Many of the turtles end up on the edge of the buildings, slightly inside. Why is this? Why does this not happen when the building colour is set to `black`? In this case, the turtles seem to have an ability to partially follow the walls of some of the buildings even though they are not applying wall following behaviour. How is this possible?

Notice what happens when the number of hills is set to 1 and the colour of the buildings is set to `black` – most of the environment ends up black. What causes this? Why does this not happen when the colour is set to `white`?

THINGS TO TRY

Try changing the values of the choosers and sliders and observe what happens in the simulation as a result.

Try changing the turtle behaviour during the simulation then observe what happens.

Try increasing the number of turtles so that at least one is created on every available space. What happens as a result for the different turtle behaviours?

EXTENDING THE MODEL

Add further turtle behaviours and/or modifications of the existing ones to the model.

NETLOGO FEATURES

To simulate the hill climbing sense, the model uses NetLogo's `uphill` command. To simulate the touch sense (for wall following via proximity detection), the model uses the `patch-right-and-ahead` command. To generate the terrain, the `diffuse` command is used.

CREDITS AND REFERENCES

Much of the code for this model is based on code provided by Uri Wilensky. It was taken from the Hill Climbing Example model and the Wall Following Example model, both available in the Code Examples directory of the Models Library.

Solution to Exercise 5.3.2: Vacuum Cleaner Robot Model

The Vacuum Cleaner Robot model provides one solution to the problem of simulating the actions of a vacuum cleaner robot. Try it out in NetLogo:

Vacuum Cleaner Robot	http://files.bookboon.com/ai/Vacuum-Cleaner-Robot.html

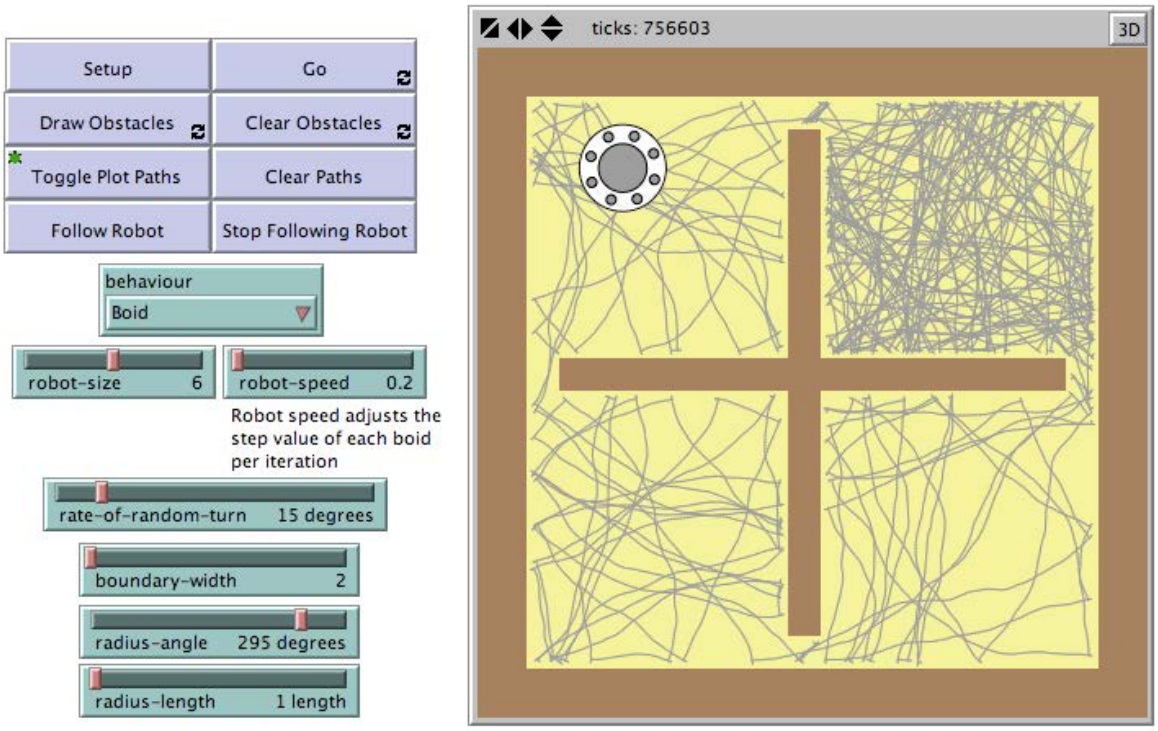


Figure S-5.3.2. Screenshot of the Interface for the Vacuum Cleaner Robot model after the setup button has been pressed followed by the go button, with the chooser and slider values as shown in the image, and further obstacles added within the environment in the form of a cross using the Draw Obstacles button.

WHAT IS IT?

This model simulates a vacuum cleaner robot whose task is to clean the floor of a room. The user can draw obstacles in the environment in order to better represent a real life environment. The model implements two basic reactive behaviours for the robot – one using a simple look ahead mechanism that reacts to any obstacles directly ahead; the other implements a boid (see Craig Reynold's work) that employs a basic obstacle avoidance steering behaviour.

This model is an extension of the Obstacle Avoidance 2 model, and part of the `Information` from that model has been duplicated here.

HOW IT WORKS

The robot turtle agent simply wanders around randomly in the environment avoiding the obstacles. The look ahead behaviour is implemented using NetLogo's `patch-ahead` command to check to see if there are any obstacles directly ahead, and if there are, it will turn a random amount. The boid behaviour is implemented using NetLogo's `in-cone` command that implements a turtle with a cone of vision.

WHAT IS ITS PURPOSE?

The purpose of the model is to show that you do not need to have a complicated method for covering the entire space of an environment. Simple reactive methods based on random walking with basic sensing will suffice.

INTERFACE

The model's `Interface` buttons are defined as follows:

- `Setup`: This sets up the environment with an outside border. One turtle agent (the robot) is created and placed at a random location.
- `Go`: The robot starts wandering around the environment avoiding obstacles.
- `Draw Obstacles`: The user can draw further obstacles in the environment. These are coloured brown.
- `Erase Obstacles`: The user can erase obstacles in the environment, including the border.
- `Toggle Plot Paths`: This instructs the robot turtle agent when wandering to put its pen down if it is up, or put it up if it is already down. If the pen is down, this will show the paths the robot takes while wandering around.
- `Erase Paths`: This will erase any paths that have been drawn in the environment.
- `Follow Robot`: This allows the perspective of the visualisation to be altered so that it is centred on the robot.
- `Stop Following Robot`: This will reset the perspective of the visualisation so that it is directly above the origin of the environment looking straight down.

The model's `Interface` chooser and sliders are defined as follows:

- `behaviour`: This sets the behaviour of the robot. There are two values:
  `"Look Ahead"`: The robot will look directly ahead to see if there is an obstacle at distance defined by the `radius-length` slider and turn a random amount if there is.
  `"Boid"`: The robot uses a basic cone of vision sense as defined by the `radius-angle` and `radius-length` sliders to determine if there are any obstacles ahead, and turns a random amount as defined by the `rate-of-random-turn` slider, with a tendency to turn to the right.
- `robot-size`: This sets the size of the robot.
- `robot-speed`: This controls the speed of the boid i.e. how much it moves forward each tick.
- `rate-of-random-turn`: This controls how much the wandering robot boid turns each time tick. The robot has a tendency to head in a right turning direction as the rate of random turn to the right (as specified by the slider) is twice that of the rate of random turn to the left.
- `boundary-width`: This sets the width of the outside boundary at the beginning when the `Setup` button is pressed. The boundary is drawn with the same colour as the obstacles (brown), therefore the robot will avoid this area as well (usually, but sometimes it can become stuck as mentioned below).
- `radius-angle`: This defines the radius angle of the boid's vision cone.
- `radius-length`: This defines the radius length of the boid's vision cone if the behaviour is set to `"Boid"` or the amount the robot looks ahead if the behaviour is set to `"Look Ahead"`.

HOW TO USE IT

Press the `Setup` button first, then press `Go`. To see where the boid wanders, press `Toggle Plot Paths`. These paths can be erased using the `Erase Paths` button.

You can draw extra obstacles by pressing the `Draw Obstacles` button and then holding down the mouse at the point where you want the obstacles to be drawn. These can also be erased using the `Erase Obstacles` if you have made a mistake. You can change the frame of reference so that the visualisation is centred around where the boid currently is situated by pressing the `Follow Robot` button. To reset the perspective, press the `Stop Following Robot` button.

THINGS TO NOTICE

Setting the behaviour to "Boid" and the robot-speed to 0.1, rate-of-random-turn to 40, radius-angle to 300, radius-length to 1, and pressing the Toggle Plot Paths button, followed by moving the speed slider (just below the Information tab in the Interface) from "normal speed" to "faster" will result in the robot boid rapidly covering the entire environment while reliably avoiding the obstacles.

Increasing the radius-length value (while keeping the other variables the same) will change how much of the space the robot covers. Instead of covering most of the environment, if the behaviour is set to "Boid", the robot will cover a space that is away from the obstacles as determined by the slider. (Note that the border is also considered an obstacle). If the behaviour is set to "Look Ahead", and the radius-length value is greater than 1, and the paths are being drawn, then this will result in gaps at the four corners of the environment if there are no obstacles inside the environment. (You will need to make the simulation go faster using the speed slider to see this more quickly). Note that if you set the radius-length slider to a higher value for the "Look Ahead" behaviour, it will often get stuck on the outside boundary depending on the width of the boundary. (Why is this?)

THINGS TO TRY

Try adjusting the robot's speed, radius angle and radius length to see how this affects the robot's behaviour. Also try changing the Interface Settings to see if this has any affect.

Try adding obstacles to see how this affects the robot's ability to cover the entire environment. For example, add obstacles in the form of a maze. Try to create "black spots" where the robot never visits. Alternatively, try to trap the robot into a small area, or try to get it stuck.

EXTENDING THE MODEL

Try adding further behaviours to the model. Observe a real vacuum cleaner robot working, and try to recreate its behaviour in this model.

The model could be extended to add gradual acceleration and deceleration. This would enhance the simulation of the robot.

NETLOGO FEATURES

The code uses the `patch-ahead` and `in-cone` commands to simulate the robot vision sense.

RELATED MODELS

See the following models: Obstacle Avoidance 1, Obstacle Avoidance 2.

Solution to Exercise 5.4.4: Mazes-2 NetLogo Model

A further maze called the Butterfly Maze has been added to the Mazes model. The model also adds two further behaviours. Try it out in NetLogo:

Mazes-2	http://files.bookboon.com/ai/Mazes-2.html

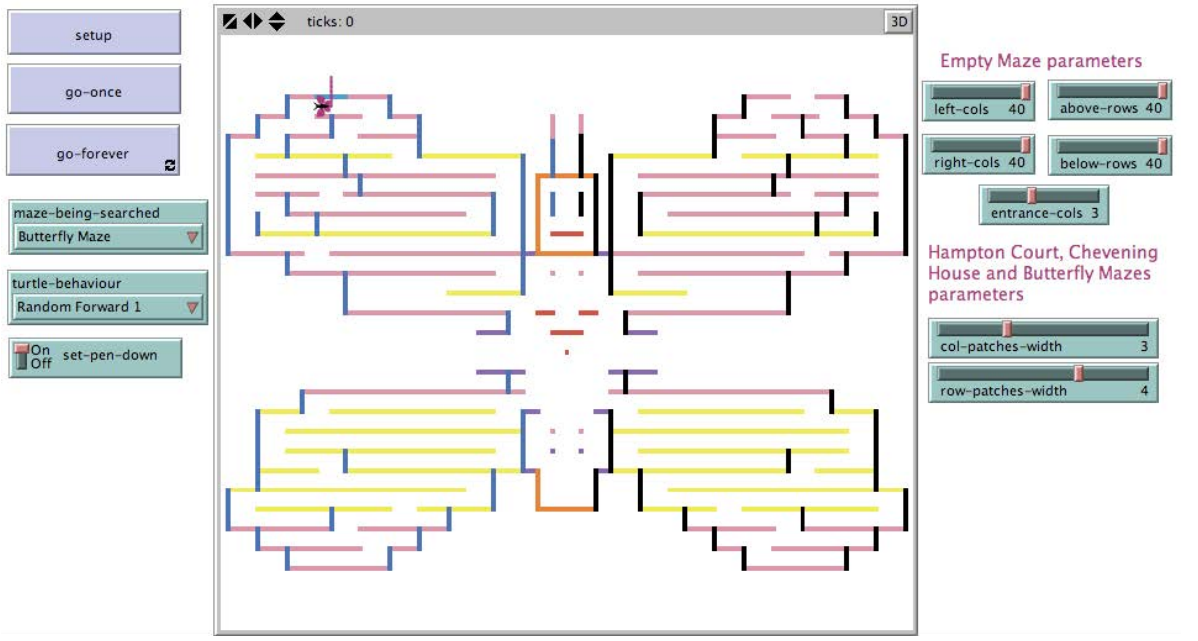


Figure S-5.4.1. Screenshot of the Interface for the Mazes-2 model after the setup button has been pressed followed by the go-forever button, with the chooser and slider values as shown in the image.

The `Information` in the model is the same as for the `Mazes` model, except for the following:

WHAT IS IT?

This model shows how to get a simple reactive turtle agent to move around a maze. The model comes with four mazes: the empty maze (just outside walls); two mazes that are schematic representations of real life mazes in the United Kingdom – the Hampton Court Palace maze, and the Chevening House Maze; and a maze that is in the shape of a butterfly, created by Jamie Hughes, a student at Bangor University in North Wales.

This model is a variant of the Mazes model, but with the Butterfly Maze added as a fourth maze. The `setup` procedure has been changed to add the fourth maze. The `setup-butterfly-maze` procedure has also been added and the `wall?` reporter has been changed to cope with different coloured walls.

The model has also added fifth and sixth turtle behaviours – `"Random Forward 3"` and `"Random Forward 4"`. These behaviours are defined by the sliders `move-forward-behaviour`, `move-forward-behaviour-after-turning` and `move-forward-step-amount`, and are the same as the behaviours used for the Searching Mazes model. This is where the turtle can perform one of three base behaviours – forward, left then forward or right then forward. The forward movement is defined by the sliders. For the `"Random Forward 3"` behaviour, the turtle chooses at random one of the three base behaviours, where the move forward action is defined by the sliders (as in the Searching Mazes model). For the `"Random Forward 4"` behaviour, the turtle chooses at random one of the three base behaviours, but ignores the slider values (except for the `move-forward-step-amount` slider) and chooses one of the four possible move forward behaviours at random (see the `The Interface` section below to see how these are defined).

THE INTERFACE

The buttons in the `Interface` are the same as for the Mazes model.

The modified choosers, and sliders are defined as follows:

- `maze-being-searched`: This specifies the maze that is being searched. This is either the empty maze, the Hampton Court Palace maze, the Chevening House maze or the Butterfly maze.
- `turtle-behaviour`: This specifies the type of reactive behaviour the walker turtle agent exhibits. The modified types of behaviour added to this model from the Mazes model are as follows:

"Random Forward 3": In this behaviour, the walker turtle agent wanders around in random directions as defined by the sliders move-forward-behaviour, move-forward-behaviour-after-turning and move-forward-step-amount, and are the same as the behaviours used for the Searching Mazes model. This is where the turtle can perform one of three behaviours – forward, left then forward or right then forward.
"Random Forward 4": This behaviour is similar to the "Random Forward 3" behaviour except that instead of the move forward behaviour being defined by the sliders, it is chosen at random.

- move-forward-behaviour, move-forward-after-turning-behaviour: These choosers control how the turtle moves forward (i.e. not turning for the first chooser, or what happens after it has turned either left or right for the second chooser) when the turtle-behaviour slider is set to "Random Forward 3" or "Random Forward 4". The options for the type of move forward behaviour used by both these choosers are as follows:

"Move forward n steps unless hit wall": The turtle moves forward a number of steps determined by the move-forward-step amount slider.

"Move forward until hit wall": The turtle will move forward indefinitely until it hits a wall.

`"Move forward until any open space"`: The turtle will move forward until it has found open space (i.e. until there is a wall in the way or there is open space on both sides).

`"Move forward until open space after side wall"`: The turtle will move forward until it has found open space (i.e. until there is a wall in the way or there is open space on both sides but will only stop if it has previously been following a wall on either side).

- `move-forward-step-amount`: This is the amount of steps forward the turtle makes when it moves forward when the `turtle-behaviour` slider is set to `"Random Forward 3"` or `"Random Forward 4"` (either directly ahead of itself on its current heading, or after it has made a right or left turn).

THINGS TO TRY

Try to find out what happens in the Butterfly Maze when different turtle behaviours are chosen. Find out what happens to the way the maze is traversed when the value of the `col-patches-width` and the `row-patches-width` sliders are altered to change the dimensions of the maze. Why does the turtle agent end up going different ways with different slider values?

THINGS TO NOTICE

The turtle agent in the Butterfly Maze often ends up running around the outside of the maze, unlike the other three mazes. i.e. It can get to the exit at the top right by cheating. The reason it does this is that there are further ways of exiting the maze in the middle that are not the final exit at the top right that constitutes the goal. This is another illustration of how an environment has an important role in determining the behaviour of an agent, since this behaviour of the turtle agent in the Butterfly Maze does not arise in the other three mazes.

Notice also that there are many “islands” in the Butterfly Maze. You can get the turtle agent executing the hand on the wall behaviour to leap from one island to another by temporarily switching the behaviour to one of the other three behaviours using the `turtle-behaviour` slider, and then switching back. How many islands are there?

EXTENDING THE MODEL

Try adding your own behaviours to get the turtle agent to move around the maze.

Solutions to Exercises 3.6.12, 4.3.1, 5.5.1, 5.5.3 and 5.5.4: Santa Fe Trail NetLogo Model

The Santa Fe Trail model provides solutions to Exercise 4.3.1, 5.5.1, 5.5.3 and 5.5.4. Try it out in NetLogo:

Santa Fe Trail	http://files.bookboon.com/ai/Santa-Fe-Trail.html

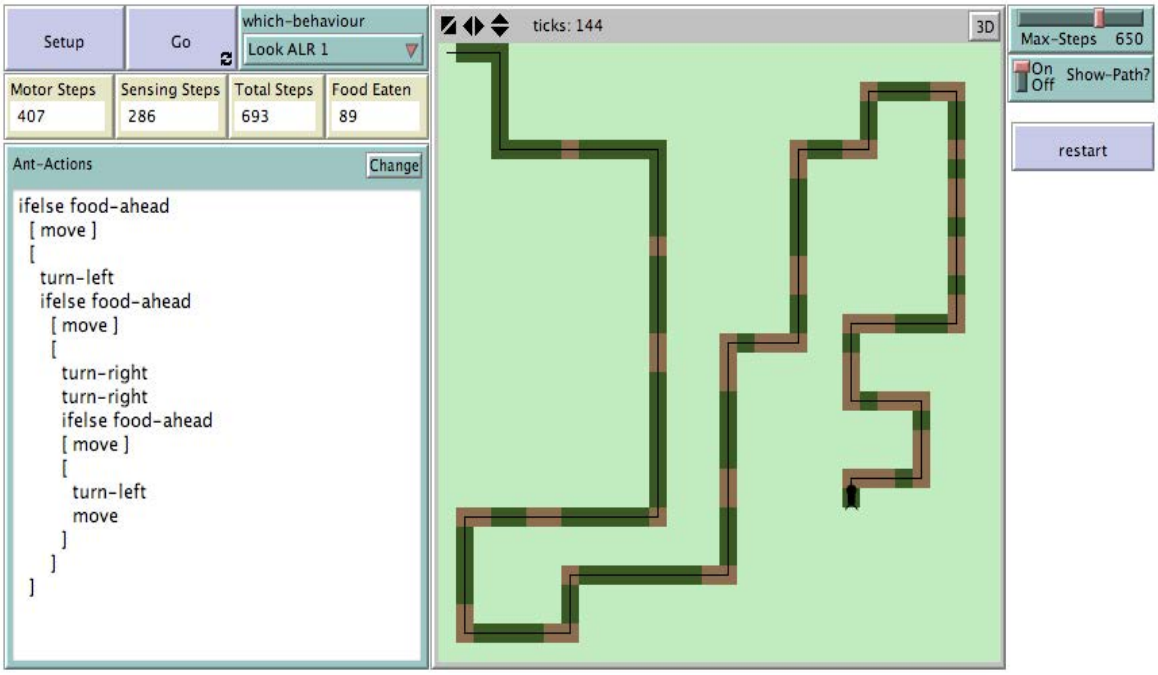


Figure S-5.5.1. Screenshot of the Interface for the Santa Fe Trail model after the Setup button has been pressed followed by the go button, with the chooser, input box and slider values as shown in the image.

WHAT IS IT?

This model tests out various behaviours as solutions to the Santa Fe Ant Trail problem. The Santa Fe Ant Trail was devised by John Koza in order to test the performance of evolutionary algorithms. This model loads the trail from a data file, then various behaviours can be loaded into the input box and modified for testing on the Trail.

HOW IT WORKS

A single ant turtle agent is created as the ant. Its starting location is at the top left of the environment. Its actions are determined by the code in the `Ant-Actions` input box of the Interface. This box is editable – the user can modify the code as she or he wishes by pressing the `Change` box. The code is then executed by the ant each tick when the `go` button is pressed.

Various motor actions for the ant are provided in the model for use in defining their behaviour and are explained as follows:

- `move`: This moves the ant forward one step.
- `turn-left`: This turns the ant left by 90 degrees.
- `turn-right`: This turns the ant right by 90 degrees.

Various sensing and/or sensory-motor actions for the ant are provided in the model for use in defining their behaviour and are explained as follows:

- `food-ahead`: This reports whether there is food directly ahead of the ant.
- `food-on-my-left`: This reports whether there is food on the left of the ant.
- `food-on-my-right`: This reports whether there is food on the right of the ant.
- `food-ALR`: This reports the action needed to be performed by the ant if there is food nearby. If there is none, then the default action is to move ahead.
- `food-direction`: This reports the direction of food if there is any nearby. If there is none, then the default direction is 0 degrees (straight ahead).

HOW TO USE IT

Choose which behaviour you want to be executed by the ant using the `which-behaviour` chooser. This will then be loaded into the `Ant-Actions` input box when the `Setup` button is pressed. Then to run the model, press the `Go` button.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `Setup`: This will clear the environment and load the Santa Fe Trail into it, and create a single ant turtle agent at the start of the trail. It will also load the code for the ant's behaviour (as specified by the `which-behaviour` chooser) into the `Ant-Actions` input box. This can then be edited by the user if they wish.
- `Go`: This will run the simulation.
- `Restart`: This will restart the simulation from the beginning without the need to press the `Setup` button (this is useful when you have edited the `Ant-Actions Input` box, since pressing `Setup` will re-load the original unedited behaviour).

The chooser, monitors, input box, slider and switch are defined as follows:

- `which-behaviour`: This specifies the behaviour that gets loaded into the `Ant-Actions` input box and subsequently executed when the `Go` button is pressed. The behaviours are defined as follows:

  `"Optimum"`: This is the optimum behaviour for the Santa Fe Trail (it follows the trail exactly).
  `"Koza"`: This is the translation from LISP to NetLogo code of the solution mentioned in Koza's first GP book (Genetic Programming: On the Programming of Computers by the Means of Natural Selection, 1992, p. 154).
  `"Koza-1"`: This was a solution discovered using Grammatical Evolution provided by Loukas Georgiou that is based on a semantically equivalent grammar to the set of programs possible within the original Santa Fe Trail definition of Koza.
  `"Koza-2"`: This is a manual modification of `"Koza-1"` to remove unnecessary steps.
  `"Koza-3"`: This is a further manual modification to `"Koza-2"` that adds a single `"move"` command in order to improve performance even further.
  `"Koza Modified"`: This is modified version of `"Koza"` which removes unnecessary steps.
  `"Look ALR 1"`: The ant looks ahead for the trail, then left, then right (no peripheral vision); it turns in the direction of food that was first found, otherwise it moves forward.
  `"Look ALR 2"`: The ant looks ahead for the trail; then left; then right (using peripheral vision). Then it moves in the direction of food.

"Look ALR 2a": This is a modification to "Look ALR 2" for a myopic ant that does not have peripheral vision (i.e. it can only use the food-ahead command to sense food as in Koza's original definition, and so it must physically turn to the left or right in order to sense the presence of food on its left or right).
"Look ALR 2b": This is a modification to "Look ALR 2a" where the ants look right first rather than left.
"Look ALR 3": This is similar to "Look ALR 2" but uses simpler code.
"Look ALR 4": This is similar to "Look ALR 2" but uses even simpler code.

- Motor Steps: This is the number of motor actions the ant performs – i.e. move, left turn or right turn.
- Sensing Steps: This is the number of sensing actions the ant performs – e.g. looking to see if there is food ahead or to its left or right.
- Total Steps: This is the sum of the Motor Steps + Sensing Steps.
- Food Eaten: This is the amount of food eaten. If the ant has managed to eat all 89 patches of food, then it completes the task.
- Ant-Actions: These are the actions the ant performs each tick of the simulation.
- Max-Steps: This is the maximum number of Motor Steps the ant is allowed to perform before the simulation is stopped.
- Show-Path?: If this is set to On, then the path the ant takes will be drawn in the environment.

THINGS TO NOTICE

Notice that the Optimum behaviour does not perform any sensing actions – there are only the motor actions needed to follow the path exactly. This is the minimum number of motor steps needed to follow the trail (hence why this is called the Optimum behaviour). However, this behaviour is too specific – it would only be useful for following the Santa Fe Ant Trail and nothing else.

Notice which of the other behaviours result in the same number of motor steps as the Optimum behaviour. How do they compare when the sensing steps are also taken into consideration?

Notice that the best solution (apart from Optimum) requiring the least number of motor steps and not using the extra sensing operators food-on-my-left and food-on-my-right, is "Koza-3". It takes 365 steps. It works better compared to the other Look ALR type solutions because it takes advantage of a quirk of the Santa Fe Trail whereby if you had to turn in order to find the trail, then you can move forward two steps rather than one and you won't lose the trail. i.e. This solution would fail on a different trail where there was a rapid change in direction.

Notice that the `Optimum` solution requires 165 steps without any sensing operations being performed. For solutions that are constrained as in Koza's original problem to physically turn left or right in order to check if there is food on your left or right, then when there is a gap in the trail where there is no food, there is a penalty involved in looking both ways and turning back again to the original heading. As there are 55 patches on the trail which have no food, then this results in a cost of 55 x 4 motor steps as the agent has to turn left, then right, then right, then left. i.e. This imposes a cost of 220 steps at least, so a minimum is 165 + 220 steps, or 385 steps.

In addition, there is the cost of turning which will add an extra 2 steps compared to the optimum for each right turn if the agent's inclination is to turn left first, and vice versa if the other way around. i.e. For left-turns-before-right-turns agents, there are 11 right turns, so added cost is 22; for right-turns-before-left-turns agents, there are 10 left turns, so added cost is 20.

This explains why some solutions discovered by evolutionary algorithms require 405 and 407 steps (385 + 20 and 385 + 22). The solutions that are less (377 motor steps for `"Koza-2"` and 365 motor steps for `"Koza-3"`) take advantage of a quirk in the trail that allows an ant to move 2 steps rather than 1 after a turn. By moving 2 steps forward, the ant jumps over some of the gaps in the trail, therefore avoiding the extra 4 steps needed that the other ants need to take.

THINGS TO TRY

Try out all the different behaviours to see which one is 'best' (i.e. uses the minimum number of total steps (motor + sensing). Compare this with the `Optimum` behaviour.

Try writing your own behaviours or modify an existing one using the `Change` button in the `Ant-Actions` input box. (For example, remove the last '`move`' command from `Koza`'s behaviour to see what happens). Make sure that you press the `Setup` button first before doing this to clear any existing paths. How well do your behaviours perform? Are they successful in getting to the end of the Trail?

EXTENDING THE MODEL

Try writing an evolutionary algorithm (via a Genetic Algorithm, Genetic Programming or Grammatical Evolution, for example) to evolve the behaviour of the ant to see what code is evolved as a result.

NETLOGO FEATURES

Note the use of the `run` command to run the commands in the `Ant-Actions` code.

RELATED MODELS

See the Follow Trail model.

REFERENCES

Jefferson, D., Collins, R., Cooper, C., Dyer, M, Flowers, M., Korf, R., Taylor, C. and Wang, A. (1991) *Evolution as a Theme in Artificial Life: The Genesys/Tracker System*. **In** Langton, Christopher, *et al.* (editors), Artificial Life II. Addison-Wesley.

Koza, J.R. (1992) *Genetic Programming: On the Programming of Computers by the Means of Natural Selection*. Cambridge, MA: MIT Press.

Langdon, W.B. and Poli, R. (1998) *Better Trained Ants for Genetic Programming*. Technical report, University of Birmingham, School of Computer Science (CSRP-98-12), April 1998.

Solution to Exercise 5.5.2: Follow Trail NetLogo Model

The Follow Trail model provides some solutions to Exercise 5.5.2. Try it out in NetLogo:

Follow Trail	http://files.bookboon.com/ai/Follow-Trail.html

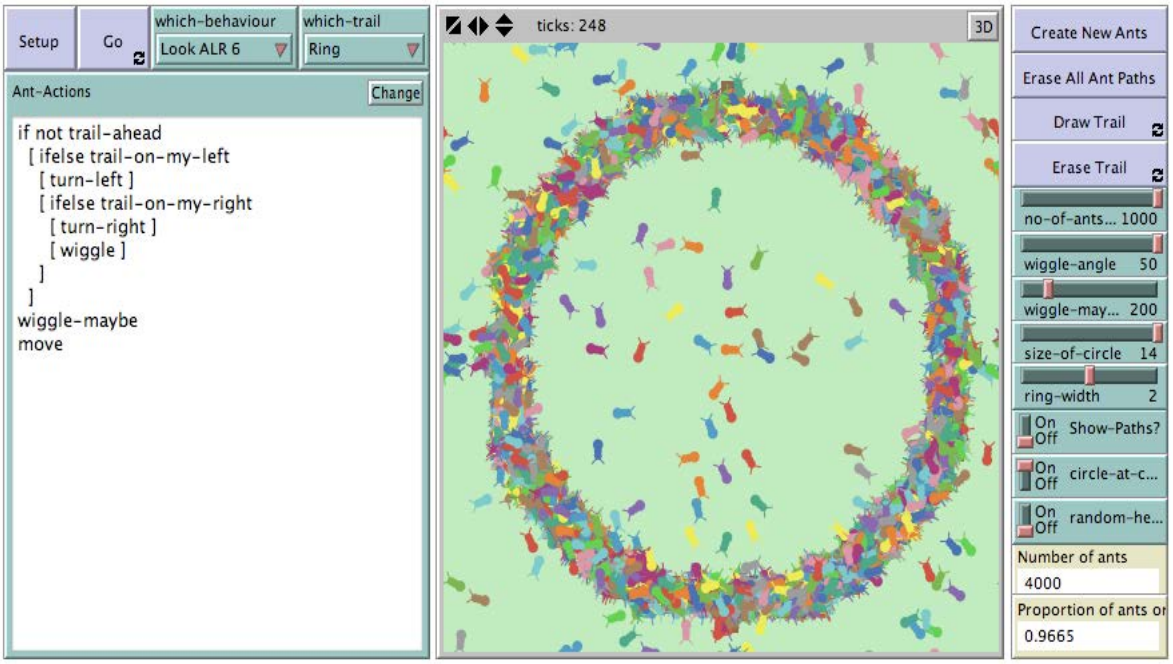


Figure S-5.5.2.1. Screenshot of the Interface for the Follow Trail model after the Setup button has been pressed followed by the go button, with the chooser, input box and slider values as shown in the image.

The `Information` that is different for this model to the related Santa Fe Trail model above is as follows. (In various places below, some repetition with the Santa Fe Trail model has been included in order to improve clarity).

WHAT IS IT?

This NetLogo model allows the user to test out various trail following behaviours for ants. It is an extension of the Santa Fe Trail model. This model allows the user to load various trails from data files, or draw them directly, then various behaviours can be loaded into the input box and modified for testing on the various trails.

HOW IT WORKS

One or more turtle agents are created as the ants at random locations. Their actions are determined by the code in the `Ant-Actions` input box of the `Interface`. This box is editable – the user can modify the code as she or he wishes by pressing the `Change` box. The code is then executed by the ant each tick when the `Go` button is pressed.

Two further motor actions for the ants to those provided in the Santa Fe Trail model are available for use in defining their behaviour. These are explained as follows:

- `wiggle`: The ant changes its direction randomly as determined by the `wiggle-angle` slider. (The amount of the change in direction will be a random number from 0 to the `wiggle-angle` value).
- `wiggle-maybe`: The ant performs a wiggle, but only with probability `1 / wiggle-maybe-ratio`.

HOW TO USE IT

Choose which behaviour you want to be executed by the ant using the `which-behaviour` chooser. This will then be loaded into the `Ant-Actions` input box when the `Setup` button is pressed. Then to run the model, press the `Go` button.

You can draw you own parts of the trail using the `Draw Trail` button, or erase parts of the trail using the `Erase Trail` button.

THE INTERFACE

The buttons in the `Interface` are defined as follows:

- `Setup`: This will clear the environment and load the trail specified by the `which-trail` slider into it, and create `no-of-ants-to-add` ant turtle agents at random locations. It will also load the code for the ant's behaviour (as specified by the `which-behaviour` chooser) into the `Ant-Actions` input box. This can then be edited by the user if they wish.
- `Go`: This will run the simulation.
- `Create New Ants`: This will add new ants into the environment, the amount being that specified by the `no-of-ants-to-add` slider.
- `Erase All Ant Paths`: This will erase all the ant paths.
- `Draw Trail`: This allows the user to draw their own trail for testing out different environments.
- `Erase Trail`: This allows the user to erase any existing parts of the trail.

The chooser, monitor, input box, sliders and switches are defined as follows:

- `which-behaviour`: This specifies the behaviour that gets loaded into the `Ant-Actions` input box and subsequently executed when the `Go` button is pressed. The behaviours are defined as follows:
- `"Koza"`: This is Koza's evolved behaviour for the Santa Fe Trail.
- `"Koza Modified"`: This is Koza's evolved behaviour for the Santa Fe Trail but modified to remove unnecessary steps.
- `"Santa Fe Optimum"`: This is the optimum behaviour for the Santa Fe Trail (it follows the trail exactly).

  `"Look ALR 1"`: The ant looks ahead for the trail, then left, then right (no peripheral vision); it turns in the direction of food that was first found, otherwise it moves forward.

  `"Look ALR 2"`: The ant looks ahead for the trail; then left; then right (using peripheral vision). Then it moves in the direction of food.

  `"Look ALR 3"`: This is similar to `"Look ALR 2"` but uses simpler code.

  `"Look ALR 4"`: This is similar to `"Look ALR 2"` but uses even simpler code.

  `"Look ALR 5"`: This is the same as for `"Look ALR 2"` but the ant wiggles if the trail is not found.

  `"Look ALR 6"`: This is the same as for `"Look ALR 5"` but `wiggle-maybe` is used to get the ant out of any loops it may be stuck in as it follows the path (e.g. such as a small square loop at trail corners).

- `which-trail`: This specifies the trail that is drawn in the environment. The trails are defined as follows:

  `"None"`: This will not draw any trail into the environment at `Setup`. User-drawn trails can be added using the `Draw Trail` button and erased using the `Erase Trail` button.

  `"Circle"`: This draws the trail as a filled in circle in the centre of the environment if the `circle-at-centre?` switch is set to `On`, and centred at the origin (top left) otherwise. (This will cause the circle to be wrapped across the four corners of the environment.) The dimensions of the circle are specified using the `size-of-circle` slider.

  `"Cross"`: This will draw the trail as a cross in the centre of the environment.

  `"Ring"`: This will draw the trail as a circle in the environment as for the `"Circle"` trail above, but the centre of the circle will be hollow i.e. it will end up as a ring shape. The width of the ring is specified using the `ring-width` slider.

  `"Santa Fe"`: This is the Santa Fe Ant Trail as for the Santa Fe Trail model.

  `"Spiral"`: This will draw the trail as a square spiral in the centre of the environment.

  `"Square"`: This will draw the trail as a square towards the outer part of the environment.

- `Ant-Actions`: These are the actions the ant performs each tick of the simulation.
- `no-of-ants-to-add`: This specifies the number of ants that are initially created at setup, and subsequently when the `Create New Ants` button is pressed.
- `wiggle-angle`: This specifies the random angle directional change if the ants employ the use of the `wiggle` or `wiggle-maybe` motor action in their behaviour.
- `wiggle-maybe-ratio`: This defines the probability that the ant performs a wiggle randomly if the `wiggle-maybe` action is included in their behaviour. The probability is set as `1 / wiggle-maybe-ratio` that a wiggle will occur.
- `size-of-circle`: This is the size of the circle when the `which-trail` chooser is set to `"Circle"`.
- `ring-width`: This is the width of the ring when the `which-trail` chooser is set to `"Ring"`.
- `Show-Paths?`: If this is set to `On`, then the path that each ant takes will be drawn in the environment.
- `circle-at-centre?`: If set to `On`, and `which-trail` is set to `"Circle"` or `"Ring"`, this will draw the circle or ring trail at the centre of the environment; otherwise, it will draw it at the origin (top left corner).
- `random-heading?`: If set to `On`, this will result in the ants having a random heading when they are created rather than heading towards the origin (top left corner).
- `Number of Ants`: This is the number of ants currently in the environment.
- `Proportion of ants on the trail`: This is the proportion of ants on the trail compared to the total number of ants.

THINGS TO NOTICE

Notice the incredible variety of situations that arise in the simulation despite the relative simplicity of both the behaviours and the trails, with many emergent phenomena occurring as a result of the interaction of the ants with the environment.

Notice that many of the ants end up getting stuck in a tight square loop, often at trail corners. There are two additional behaviours added to this model from the Santa Fe Trail model (`"Look ALR 5"` and `"Look ALR 6"`) that include some random 'wiggling' to provide the ants with a random element to their behaviour. However, notice that the ants' ability to follow the trails for the `"Look ALR 6"` behaviour appears to be slightly compromised as a result compared to the other similar `"Look ALR"`-type behaviours that do not include the addition of the `wiggle-maybe` action prior to the final `move` action.

Notice that the `Santa Fe Optimum` behaviour does not perform any sensing actions – there are only the motor actions needed to follow the Santa Fe Trail exactly. This is the minimum number of motor steps needed to follow the trail (hence why this is called the `Santa Fe Optimum` behaviour). However, this behaviour is too specific – it is only useful for following the Santa Fe Ant Trail and nothing else. For example, if you set the `which-behaviour` slider to `"Santa Fe Optimum"` and the `which-trail` slider to `"Santa Fe"`, you will notice that unless the ant is located at the start of the trail and pointing in the right direction at setup, it will not follow the trail at all. (The paths of all the ants do however create an interesting pattern).

Notice what happens when you select the `Circle` trail, set the number of ants to maximum, and turn off the `Show-Paths?` switch. The ants seem to spin around the circle. At the beginning of the simulation, they also head en masse to four different quadrants of the circle if the `random-heading?` switch is set to `Off`.

Notice what happens when you turn the `random-heading?` switch to `Off`. The ants for most behaviours will head in clumps to a common point (the origin, or top left corner) at the beginning of the simulation before they acquire the trail. This can lead to some interesting effects for some of the trails. Try repeatedly adding more ants into the environment when you are testing this out.

Notice which behaviours in relation to which trails lead to higher proportions of ants on the trail (as measured by the `Proportion of ants on the trail` monitor).

THINGS TO TRY

Try out all the different behaviours on the different trails to see which do well at keeping to the trail. Which trails cause problems for the agents to follow?

Try out different values for the sliders to see what affect this has on the ants' abilities to follow the trails. Try out with a small number of ants and then with a large number of ants (by changing the value of the `no-of-ants-to-add` slider). Turn the `Show-Paths?` switch on and off to see the simulation with and without the paths drawn. Try adding more and more agents by repeatedly pressing the `Create New Ants` button. (You will need to turn the `Show-Paths?` switch to `Off` when you do this, otherwise the ants' paths will quickly swamp the visualisation).

Try writing your own behaviours or modify an existing one using the `Change` button in the `Ant-Actions` input box. (For example, remove the last `'move'` command from `Koza`'s behaviour to see what happens). Make sure that you press the `Setup` button first before doing this to clear any existing paths. How well do your behaviours perform?

Try drawing a one-patch wide cross inside of the ring trail when the `which-behaviour` slider has been set to `"Look ALR 5"` after the simulation has been running for a while so that the `Proportion of ants on the trail` is at 1 (i.e. 100% of the ants are on the ring). Why do the ants fail to follow the inside cross? Next try increasing the width of the cross to two patches or more. What happens now and why?

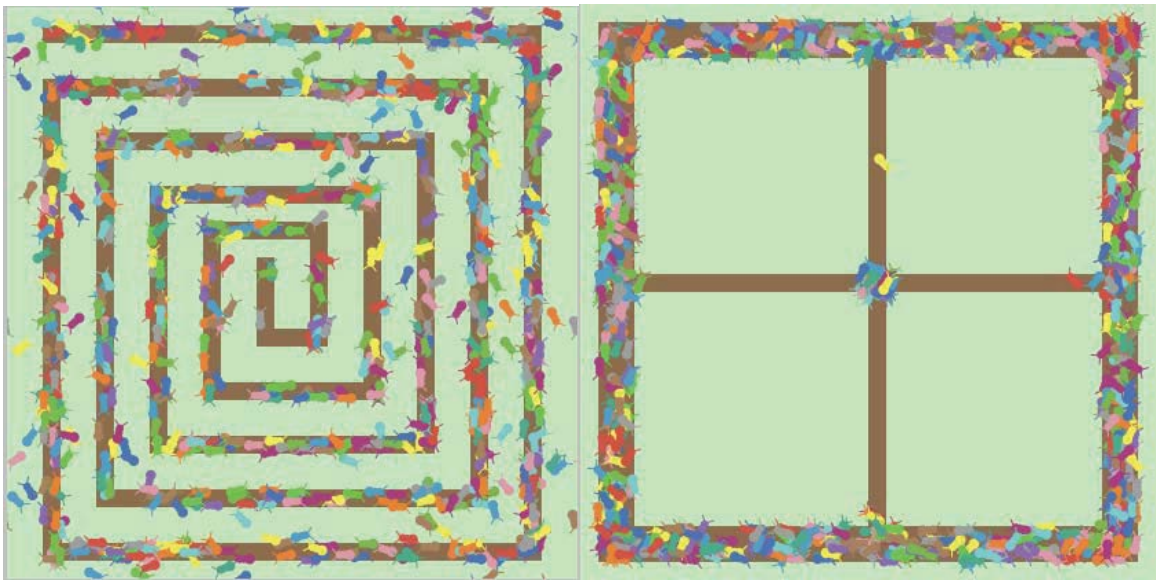


Figure S-5.5.2.1. Further screenshots from the Follow Trail model.

EXTENDING THE MODEL

Try modifying the behaviour to include collision detection so that the ants avoid other ants heading their way. Also add in a crowd following behaviour where the ants head in the direction of the crowd when there are a lot of other ants heading the same way.

RELATED MODELS

See the Santa Fe Trail model.

Solution to Exercise 5.5.3:

One solution is to measure the proportion of ant agents that are on a trail patch – the higher the proportion, the more effective the behaviour at self-organising the ants to follow the trail. The following NetLogo reporter will return this proportion as a number from 0 to 1:

```
to-report proportion-of-ants-on-trail
   report (count turtles with [pcolor = colour-of-trail] /
           count turtles)
end
```

You can find the reporter included in the code for the Follow Trail model described in Solution to Exercise 5.5.2. The monitor `Proportion of ants on the trail` displays the value returned by the reporter in the `Interface`.